CONSTRUCTING COST-EFFECTIVE AND TARGETABLE ICS HONEYPOTS SUITED FOR PRODUCTION NETWORKS

THESIS

Michael M. Winn, Major, U.S. Army

AFIT-ENG-MS-15-M-045

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-15-M-045

CONSTRUCTING COST-EFFECTIVE AND TARGETABLE ICS HONEYPOTS SUITED FOR PRODUCTION NETWORKS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Cyber Operations

Michael M. Winn, B.S.

Major, U.S. Army

26 March 2015

AFIT-ENG-MS-15-M-045

CONSTRUCTING COST-EFFECTIVE AND TARGETABLE ICS HONEYPOTS SUITED FOR PRODUCTION NETWORKS

THESIS

Michael M. Winn, B.S.
Major, U.S. Army

Committee Membership:

LTC Mason Rice, PhD (Chairman)

Barry E. Mullins, PhD (Member)

Juan Lopez, Jr. (Member)

Abstract

Our nation's privately owned critical infrastructure is at risk. Malicious actors are constantly developing exploitation techniques that circumvent traditional IP controls. Honeypots are a technique that can mitigate the risk of these threats. Effective honeypots are authentic and targetable, and their design and implementation must accommodate risk tolerance and financial constraints. Developing honeypots for Industrial Control System (ICS) networks is difficult. The proprietary, and often expensive, hardware and software used by ICS devices creates the challenging problem of building a flexible, economical, and scalable honeypot.

Honeyd is an existing open source framework designed to create a large footprint of traditional Information Technology (IT) honeypots, however, it lacks authenticity when implemented with an ICS. This research extends Honeyd into Honeyd+, making it possible to use the proxy feature to create multiple high interaction honeypots with a single Programmable Logic Controller (PLC).

This research begins with pilot studies to examine the behavior of the PLC and Honeyd. The results establish scope and context for the development and evaluation of the Honeyd+ modifications. Honeyd+ is tested with a network of 75 decoy PLCs, and the interactions with the decoys are compared to a physical PLC to test for authenticity. The performance test evaluates the impact of multiple simultaneous connections to the PLC to determine targetability.

The functional test is successful in all cases. The performance test demonstrates that the PLC component is a limiting factor, and that introducing Honeyd+ has a marginal impact on performance. Notable findings are that the Raspberry Pi is the preferred hosting platform for the EtherNet/IP protocol and more than five simultaneous connections significantly decrease performance.

Acknowledgements

I would like to thank my committee for their guidance, mentorship, and assistance throughout this research process.

Stephen Dunlap deserves special recognition for his assistance with setting up the lab gear, troubleshooting code, and honest advice in this endeavor.

I would like to thank my lovely wife and children for supporting me with all of the reading sessions, late nights in the lab, and the occasional frustration when the project was especially challenging.

Michael M. Winn

Table of Contents

Page

List of Figures

Figure Page

List of Tables

CONSTRUCTING COST-EFFECTIVE AND TARGETABLE ICS HONEYPOTS SUITED FOR PRODUCTION NETWORKS

I. Introduction

1.1 Motivation

On November 20, 2014, the Director of the National Security Agency, Admiral Mike Rogers stated that several actors, including China, Russia, and other groups have the capability to disrupt utilities such as the distribution of electricity and other energy assets throughout the United States causing physical destruction, personal injury, and even death [10].

Admiral Rogers further expressed his desire to share threat intelligence with the private sector, but implied that the private sector lacks the ability to collect and analyze precise network sensor data required to make the intelligence useful in detecting, preventing, and recovering from a cyber attack. This is due in part to reliance on IP-based protection devices, such as intrusion detection systems and firewalls. Further compounding the problem, malicious actors have a fundamental understanding of current signature-based sensor technologies and can engineer malware to elude such systems, making their activities undetectable. A honeypot is an effective method to learn about an attacker's tactics, techniques, and procedures [25, 31].

Cyber threats targeting critical infrastructure are genuine, and the upward trend of incidents poses significant risk to national security. In 2013, the Department of Homeland Security (DHS) Industrial Control System Cyber Emergency Response Team (ICS-CERT) responded to 257 cyber incidents. The energy sector, which in-

cludes petroleum, natural gas, and electricity generation and distribution, accounted for 145 (56%) of those incidents. Cyber incidents on critical infrastructure continue to increase, up 30% from the previous year [11].

1.2 Assumptions

This research uses the following assumptions:

- Traditional IT honeypot techniques do not transfer or scale easily to the ICS sector. Hardware and software components are often proprietary and have little publicly available documentation, which hinders third party development of protection mechanisms.
- Equipment cost constrains honeypot implementation on Industrial Control System (ICS) networks.
- Computing resources for ICS components are designed to serve a specific purpose, and are focused on accuracy and response time. ICS components do not have the extensive processing power and memory resources of general purpose computers that are designed to run multiple applications at once.
- Private organizations operating critical infrastructure are reluctant to introduce additional mechanisms to their control systems. This is due to perceived–or real–risk of interference, exposure of competitive business practices, or lack of funding.

1.3 Research Goals and Hypothesis

The overarching goal of this research is to extend the functionality of an existing honeypot framework so that it can replicate many decoy PLCs using one physical PLC. The research hypotheses are:

- A passive network component between the attacker and the physical PLC can present an accurate representation of many decoy PLCs to the attacker.
- The performance limitation of a PLC is the limiting factor of performance in the honeypot implementation, thus limiting targetability.
- The high cost platform hosting the decoys has more computing resources available and will outperform the low cost platform.

1.4 Related Research

This research extends the work of R. Bodenheim, where a honeypot implementation examined whether the Shodan search engine lead to increased cyber attacks on ICS devices exposed to the public Internet [3].

1.5 Thesis Layout

Chapter 2 describes the background and establishes context for this research. Chapter 3 describes the development of the honeypot, and Chapter 4 explains the methods used to evaluate the honeypot. Chapter 5 presents the results of the evaluation. Finally, Chapter 6 offers conclusions and opportunities for further research.

II. Background

This chapter provides the background knowledge needed for a basic understanding of honeypots, deploying honeypots in an ICS environment, and a description of the existing Honeyd framework. A basic understanding of networking and ICS environments (e.g., components, basic functions, architecture, and terminology) is assumed. Section 2.1 defines basic characteristics of honeypots, providing the basis for evaluation. Section 2.2 provides a brief background on traditional honeypot terms and technologies to establish context. Section 2.3 focuses on ICS honeypots, including previous research. Finally, a description of the features and functionality of Honeyd is introduced in Section 2.4.

2.1 Honeypot Characteristics

Successful honeypots balance the following characteristics: authenticity, targetability, cost, and risk. An authentic honeypot mimics the features of a real computer system. More realistic features yield a more complex honeypot, and collects more detailed data. Honeypots are useless if the attacker avoids them. Therefore, a honeypot must be targetable; displaying a large enough presence to attract the attacker's attention. The financial cost is a major consideration, and includes the initial development, deployment, maintenance, operation, data storage, and analysis of the data collected.

Finally, there are several risks associated with a honeypot, namely attracting an attacker, and dealing with the consequences of an attacker discovering the honeypot– disabling or revealing the existence of the honeypot to others. One should carefully consider risks, consequences, and mitigation measures prior to implementing any honeypot solution.

Balancing cost with size and authenticity results in design tradeoffs that focuses the honeypot implementation goals on the various stages of a cyberattack: reconnaissance, enumeration, gaining initial access, and maintaining access [29]. Honeypots that are minimally authentic are suitable for detecting automated reconnaissance probes, while high authenticity honeypots are required to study highly complex attacks, such as an advanced persistent threat (APT).

2.2 Traditional IT Honeypots and Honeynets

Research on IT honeypots is plentiful and continues to evolve. Prior to 2006, traditional high-interaction honeypots were limited by costly physical hardware. Several research efforts, including The Honeynet Project [30], explored the problem of creating and implementing realistic (i.e., authentic) honeypots with meaningful data collection mechanisms, while minimizing the costs to develop, deploy, and maintain. Virtualization technologies became widespread in the IT market between 2006 and 2008, making it convenient and inexpensive to deploy multiple high-interaction virtual machines as honeypots to collect detailed data on cyber threats [25].

Some honeypots, called production honeypots, are created to serve as decoy systems. The decoys obscure the valuable assets, while logging mechanisms such as `syslog` monitor for anomalous activity. A honeynet is an implementation of multiple high-interaction honeypots in an engineered architecture that closely mimics the aspects of a real production network. Deploying and monitoring a honeynet involves increased financial costs, in the form of system design, equipment requirements, and personnel considerations for managing the collection, analysis, and overall operation of the honeynet [31]. There is no standard size of a successful honeynet. The security objectives, network architecture, and available resources of the asset owner influences the quantity of decoy systems required for an effective honeynet.

In a honeynet architecture, the individual honeypot components should appear and function identically to their real counterparts so that the attacker cannot distinguish the real systems from the decoys. However, the costs for such authenticity multiplies as the size (i.e., targetability) increases, quickly making this solution prohibitive. In other words, honeynets have a footprint big enough to attract potential attackers, but the complexity of programming and equipment required can result in lesser authenticity than what is required to capture and maintain the attention of a serious malicious actor.

2.2.1 Production Versus Research Honeypots.

There are generally two categories of honeypots: production honeypots and research honeypots. Production honeypots are primarily intended as sensors or decoys to supplement existing security mechanisms on a production network [31]. Since they are deployed on a production network, the goals are detection and deception; keeping the attacker engaged with the honeypot so that they will not attack other assets on the network. The ultimate goal is to use the production honeypot as a tool to supplement signature-based detection devices in quickly detecting and recovering from an attack. Production honeypots focus on minimizing costs and risk to the existing network.

Research honeypots are intended to attract an attacker and sustain the attack for as long as possible for the purpose of collecting detailed information such as unforeseen attack vectors, zero-day exploits, or post-attack activity (e.g., search terms for sensitive data, production of phishing email messages, or capturing the attacker's key strokes). While this information is not directly valuable to the average business organization, it helps security researchers develop an understanding of the attacker, their methods, and their motivations [30]. Research honeypots must be highly authentic,

so they are generally more complex to configure and involve more effort to deploy, monitor, and analyze.

2.2.2 Levels of Interaction.

There are generally two levels of interaction to describe the authenticity of a honeypot. Low interaction honeypots emulate a limited behavior of a small number of services. For example, a low interaction honeypot could return a static HTML file in response to an HTTP GET request on TCP port 80. Low interaction honeypots can range from simple scripts to elaborate emulation programs that require little configuration, resources, or effort to deploy and manage, particularly in great numbers. For these reasons, low interaction honeypots are an attractive foundation for production honeypots [25]. Since low interaction honeypots only expose a subset of exploitable features, they are limited in the amount of information they can collect. The reduced authenticity also contributes to reduced risk. An attacker interacts with a controlled simulation of predetermined responses and behaviors; a carefully designed low interaction honeypot is less likely to become assimilated into a botnet or used as a pivot than a real system.

High interaction honeypots expose the full functionality of an operating system and applications. Since it can be compromised in unexpected ways, additional mitigation techniques are required to isolate the malicious activity and reduce the risk to the production network. High interaction honeypots often require extensive configuration and dedicated equipment to ensure they are believable to an attacker. Common techniques include rigging them with specific vulnerabilities, default passwords, and phony sensitive data files. Covert monitoring logs all activity on the honeypot without the attacker being able to detect that their activities are being recorded [25]. The advantages and disadvantages of the two levels of interaction are summarized in

Table 2.1.

Table 2.1. Advantages and disadvantages of different levels of interaction [25].

Interaction	Uses	Advantages	Disadvantages
Low	Production	Lower cost Lower risk Large presence potential	Limited authenticity
High	Research	Highly authentic	Higher cost Higher risk Presence constraints

2.3 Industrial Control Systems (ICS) Honeypots

The ICS sector lacks the advantages of virtualization that the IT sector has, so there are currently few honeypot options that offer the same level of functionality and flexibility as for traditional IT systems. Current research on ICS honeypots consists mostly of low-interaction emulators [22, 26]. Low-interaction honeypots can be deployed on inexpensive, flexible platforms like a Raspberry Pi or Gumstix, which is a low-cost approach to replicating multiple decoys. One hidden drawback to low-interaction honeypots is the extensive time and labor required to develop vendor-specific protocols and usually feature a limited set of capabilities (e.g., lack of authenticity, small size, etc.) [2, 16].

Research with high interaction ICS honeypots demonstrates more flexibility and the potential to collect detailed data on evolving attack tactics, techniques, and procedures. The investment in a full-scale test bed is prohibitive to all but well-funded research institutions and governments [15]. Such full-scale systems are rare and their use is limited to controlled experimentation and research. They are rarely exposed to the Internet to collect live threat data. Furthermore, these test bed environments are often built to model a specific environment (e.g., power grid, water treatment plant,

or refinery) and not flexible enough to adapt to implementation across a diverse industry.

A single Programmable Logic Controller (PLC) can range in cost from several hundred dollars, such as the OMRON CP1L (approximately $900), to several thousand dollars, such as the Allen-Bradley 1756-L61 (approximately $5500). The cost of multiple physical devices quickly limits the scalability for implementation as expendable honeypots [32]. For instance, deploying authentic, targetable honeypots consisting of 30 L61 PLCs, the cost may be upward of $165,000 for the initial hardware.

2.3.1 Related Research.

CONPOT.

The Honeynet Project spawned CONPOT in May 2013 as an open source ICS honeypot development effort [26]. The python-based framework features a configurable emulator (i.e., low-interaction) that mimics a Siemens S7-200 PLC (MODBUS, HTTP, SNMP, and s7comm) and a web-based HMI. CONPOT encourages volunteers to adapt and deploy instances of the emulator worldwide, sending their results back to members of the project for analysis [26]. CONPOT demonstrates a decentralized method of achieving a large honeynet by distributing the cost and risk among many volunteers. This approach may not be acceptable for private enterprises, who may interpret sharing such information as a threat to their competitive business strategy.

Impact of the SHODAN Computer Search Engine.

In 2013, R. Bodenheim conducted a study with physical PLC honeypots (i.e., high interaction) to determine if discovery by the Shodan [19] search engine increased attack activity [3]. His conclusions noted that the honeypot design was limited to four devices, and expanding the size of his honeypot (e.g., the number of honeypots

exposed) was an opportunity for further research. One of Bodenheim's recommendations was to amplify the presence of one physical PLC with the proxy capability of a tool called Honeyd, hosted on a cloud computing platform such as Amazon's EC2 service [3].

2.4 Honeyd

Honeyd is a low-interaction honeypot framework, developed by Neils Provos in 2004. Honeyd is free, open source software, released under GNU General Public License[24]. It was designed to simulate network behavior and topologies for the purpose of studying and defeating Internet worm propagation, such as the well-known Slammer [6, 21], Code Red [5, 20], and Blaster [1, 7] worms .

The features of Honeyd make it an ideal framework on which to begin creating a high-interaction honeynet that avoids acquiring dozens of physical PLCs. Namely, it can create a large footprint and it is freely available. The desired honeypot is protocol independent to accommodate a wide range of ICS equipment, and can achieve the size required for effective production honeypot implementations.

The central component of Honeyd is a plain-text configuration file in which host templates are defined. A Honeyd template defines the characteristics that the honeypot host displays to the attacker. Honeyd hosts are created by associating IP addresses with a template using the keyword "bind."

The main characteristic of a Honeyd template is a "personality", which is also the keyword used to identify the operating system for the template to mimic. Honeyd personalities are an implementation of the `nmap` fingerprint database, `nmap-os-db`[18]. Honeyd can also generate the Media Access Control (MAC) address. With a specified name, Honeyd matches the vendor portion of the MAC address from the `nmap` implementation of the IEEE Organizationally Unique Indentifiers (OUI) table [18], and

can generate the remaining host portion. The Ethernet address can also be explicitly specified, if desired.

Other characteristics of a template include ICMP type, TCP ports, and UDP ports. The essence of the low-interaction capability provided by Honeyd is the response, or "action," carried out by associating scripts, executables, or subsystems with each port. Honeyd features a built-in proxy capability that can forward incoming connections to a physical `host:port`, providing a potential for high-interaction functionality [23]. Figure 2.1 illustrates the basic architecture for a proposed implementation of Honeyd, and the interaction between an attacker and the PLC.

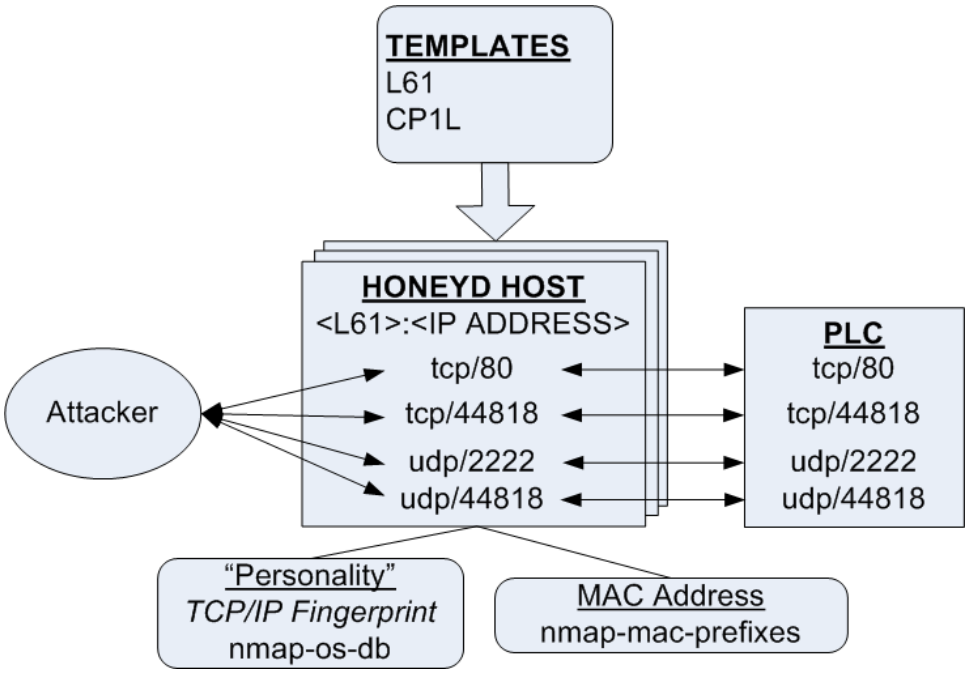


Figure 2.1. Anatomy of a Honeyd host.

III. Device Description

This chapter covers definitions, design criteria, and development of the Honeyd honeypot. Definitions and design considerations are discussed in Section 3.1. The pilot studies are described in Section 3.2, and the subsequent modifications made to Honeyd are addressed in Section 3.3.

3.1 Design Considerations

The goal of this research is to create an ICS honeypot that increases targetability and authenticity without the increased cost. Honeyd is selected as the underlying platform for the honeypot due to its demonstrated ability to generate a large footprint of decoys [24]. The two PLCs used to conduct exploratory testing are the Omron CP1L with a CP1W-EIP61 EtherNet/IP adapter and Allen-Bradley 1756-L61 ControlLogix with a 1756-EWEB module.

3.1.1 Targetability.

A targetable honeypot should consist of multiple decoys to attract an attacker's attention among the millions of other hosts on the Internet. The decoys should also appear in relevant surroundings. A lone PLC on the Internet is of little significance to an attacker, because it is unlikely to be a part of a larger system. Several PLCs on a contiguous subnet, perhaps with additional components such as a Human Machine Interface (HMI) computer, could convince the attacker that the decoy is a legitimate control system [4]. The proposed Honeyd implementation proxies several decoys to a single PLC, therefore the PLC must be able to support higher data rates over multiple simultaneous connections. One approach to measure the performance of the PLC over various data rates is by error rate, or the ratio of sent data to valid responses. Optimal

performance consists of an error rate below an established threshold.

3.1.2 Authenticity.

The decoy should be convincing enough that an attacker—either human or automated—could progress through the reconnaissance, enumeration, and initial access phases of an attack [29]. A common tactic during the reconnaissance and enumeration stages is to use a port scanning and fingerprinting tool. Once the attacker observes the decoy, the next step is to explore it.

`Nmap` is an open-source and freely available tool that can identify hosts on a subnet (i.e., by IP address) and enumerate the open TCP/IP ports on each host. `Nmap` has the capability to determine the host operating system–called fingerprinting–based on analyzing the responses received from sending TCP/IP probes to open and closed ports on a host. `nmap` has a feature called the Nmap Scripting Engine (NSE) that can perform additional enumeration tasks, such as vulnerability detection in the form of external scripts [18].

After identifying the hosts, and enumerating the operating system and network services running on each of the hosts, the attacker's next step is to explore the services for additional information and exploitable vulnerabilities. The default `nmap` settings enumerates the system ports (TCP/UDP 1-1023). The result of an initial scan of a PLC would show the common TCP port 80/http as open. Exploration of the web page would confirm that the device is a PLC, and could lead to additional protocol-specific reconnaissance on user ports (TCP/UDP 1024-49151). Another consideration is consistency, which is discussed in Section 3.2.2.

Measuring authenticity is accomplished by reproducing an attacker's methods with available tools, observing the responses of various commands and inquiries, under both normal and abnormal conditions. Successful authenticity consists of observing

accurate and consistent responses from the decoy that a similar, real physical device would produce.

3.2 Pilot Studies

A series of pilot studies is required to determine the feasibility of using Honeyd as an ICS honeypot solution, and to shape the specific metrics for evaluating authenticity and targetability. The first pilot study enumerates the network identity of two PLCs. The results form the basis for the initial configuration of Honeyd as well as a baseline for comparison. The second pilot study compares an enumeration of a Honeyd host to the baseline to determine if there are observable authenticity gaps. The third pilot study establishes the range of data rates to develop a quantifiable metric for targetability.

3.2.1 Pilot Study 1: Building a Honeyd Template.

The first pilot study uses `nmap` with operating system fingerprinting and an exhaustive enumeration of the full range of TCP/UDP responses of two different PLCs. The `nmap` options used are summarized in Table 3.1.

Table 3.1. `nmap` options used to exhaustively enumerate both PLCs

Option	Description
`-sS`	TCP SYN Scan
`-sU`	UDP Scan
`-p*`	Scan TCP and UDP ports 1-65535
`-PE`	ICMP Echo probes
`-PP`	ICMP Timestamp Probes
`-A`	Enable OS detection, version detection, script scanning, and traceroute
`-sC`	Runs all of the NSE scripts in the "Default" category (101 scripts total)
`-T4`	Aggressive timing
`-vv`	Very verbose output

The results, summarized in Table 3.2, are used to construct the Honeyd templates. Verbose results are in Appendix A. For a more efficient approach, only the well-known ports (1-1024) and the additional discovered ports (e.g., 2222 and 44818) are included in subsequent `nmap` scans. This is a safe assumption because `nmap` requires at least one open and one closed port to conduct fingerprinting, and additional ports are unlikely to appear on either the PLC or the Honeyd platforms without intentional configuration.

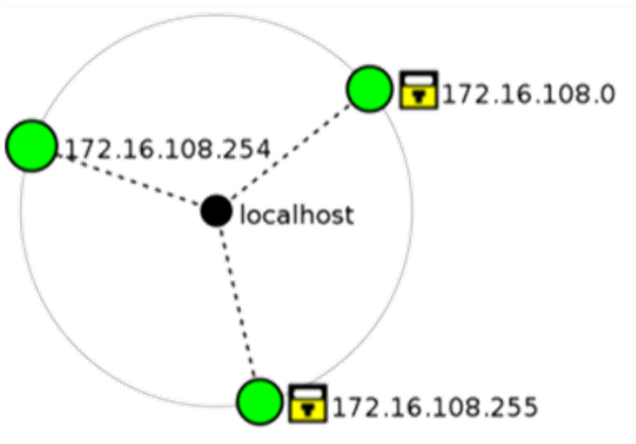
Table 3.2. Pilot study 1 results.

	L61	**CP1L**
OS	VxWorks	HP iLO 2 remote management interface
MAC Address	00:1D:9C:BE:67:93 (Rockwell Automation)	00:1D:4B:F0:22:66 (Grid Connect)
Open Ports	tcp/80 tcp/44818 udp/2222 udp/44818	tcp/80 tcp/9999 tcp/44818 udp/69 udp/2222 udp/30718 udp/44818

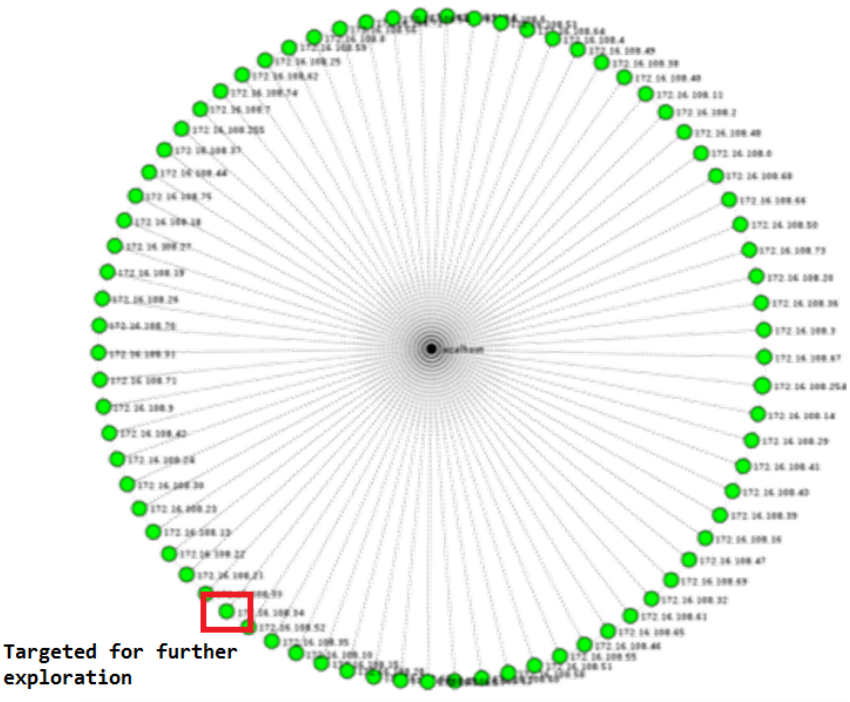
3.2.2 Pilot Study 2: Honeyd Authenticity Study.

Figure 3.1 illustrates an `nmap` scan of an empty subnet, and the same subnet after launching Honeyd configured with a template of 75 Allen-Bradley 1756-L61 ControlLogix PLCs, using the information from pilot study 1. In the figure, each dot is labeled with the IP address that responded to the `nmap` probes.

The initial test of Honeyd with an `nmap` scan appears successful. `Nmap` correctly identifies the operating system and the open ports. However, upon deeper examination of the Honeyd hosts, authenticity flaws are discovered. Each of the Honeyd hosts are identical, with no uniqueness among them (see Figure 3.2). A request for



Before: 172.16.108.0/24
Not running Honeyd



After: 172.16.108.0/24
Honeyd with an L61 template of 75 IP Addresses

Figure 3.1. The attacker's perspective: results after running an nmap scan on an empty subnet (top) with 75 Honeyd hosts (bottom).

the web page from all of the Honeyd hosts results in the same static web page for the physical PLC, MAC Address, hostname, and serial number. Additionally, the IP address listed on the web page does not match the IP address used to connect to the Honeyd host. Pilot study 2 establishes the qualitative attributes of authenticity that will be used in the evaluation. The successful attributes are denoted with a ✓in Table 3.3. Missing attributes are denoted with an **X**.

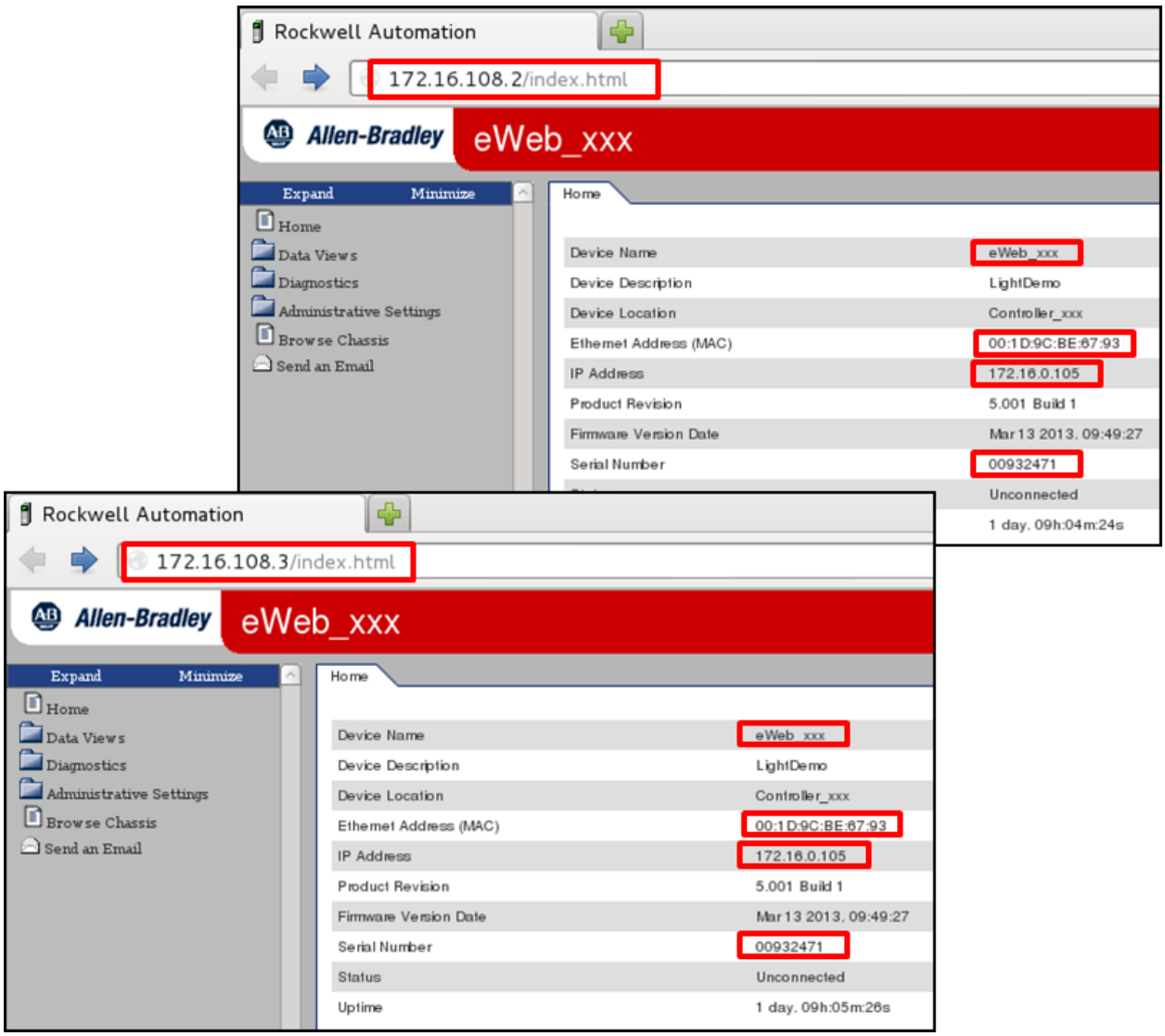


Figure 3.2. Authenticity flaws with the Honeyd hosts.

Table 3.3. Results of pilot study 2.

Metric	Consistent	Unique
Open Ports	✓	✓
OS Fingerprint	✓	✓
IP Address	**X**	**X**
MAC Address	**X**	**X**
Hostname	**X**	**X**
Serial Number	**X**	**X**

3.2.3 Pilot Study 3: PLC Performance Study.

A custom Python script is designed to test performance of both the EtherNet/IP protocol [9] and the HTTP protocol on each PLC. The script uses the python `ENIP` class developed by S. Dunlap to craft the necessary packets for communication with a PLC [12]. The test begins by opening a TCP connection to the PLC or Honeyd host, and sending a protocol-specific request. It asynchronously collects and evaluates the response, and the process is repeated for a fixed duration. Connections are executed in separate execution threads in order to simulate simultaneous connections. At the end of the test duration, the TCP connections are closed gracefully and the script outputs the results to a text file. The error rate metric (lower is better), is calculated from the output. The test script code is included in Appendix B.

A third pilot study validates the accuracy of the performance testing script and establishes a maximum effective data rate for further testing. Testing conducted over 10, 30, and 60 second durations for each collection period showed that 30 seconds was sufficient to reach a steady state on both PLC platforms considered for this study. The performance pilot study also revealed that as the sending data rate increases, the PLC responds by reducing the TCP Window size, eventually to zero. Thus, establishing an error rate based on $(send_{success} - receive_{success})/send_{success}$ will result in measurements biased by latency. For this research, the error rate definition is slightly modified by

fixing the number of requests sent in a fixed time period to reduce measurement bias related to latency. Therefore, considering the number of attempted requests against the number of valid responses removes the bias. See Equation 3.1.

$$ErrorRate = \frac{(send_{attempt} - send_{success}) + (send_{success} - receive_{success})}{send_{attempt}} \tag{3.1}$$

Finally, the pilot study showed that the maximum number of simultaneous EtherNet/IP Sessions was 64 for the L61, and only 4 for the CP1L. Testing the maximum sessions consisted of iteratively creating a TCP connection, followed by registering an ENIP session. Both PLCs stopped responding to the `register_session` request when they reached their maximum. The connection limitation is documented in the L61 specifications [27]; the limitation is not listed in either the CP1L nor the CP1W EtherNet/IP module specifications [13]. The limitations apply to the maximum number of simultaneous EtherNet/IP sessions that the PLC can handle at any given point in time, however, it does not impact the ability for Honeyd to advertise hosts.

An error rate threshold of 25% is selected because it is the approximate point where both PLCs begins reducing the TCP window size, resulting in increased error rates (see Figure 3.3). The results of the pilot study establishes reasonable data rates, shown in Table 3.4.

The TCP connections for the HTTP protocol timed out (e.g., RST) if no data was sent, and the PLC closed the connection gracefully (e.g., FIN, FIN-ACK, RST) at the conclusion of each reply. It should be noted that the error rates of the web servers on both PLCs was substantially higher than EtherNet/IP, particularly on the CP1L, where the lowest error rate was approximately 60% (e.g., above the 25% threshold).

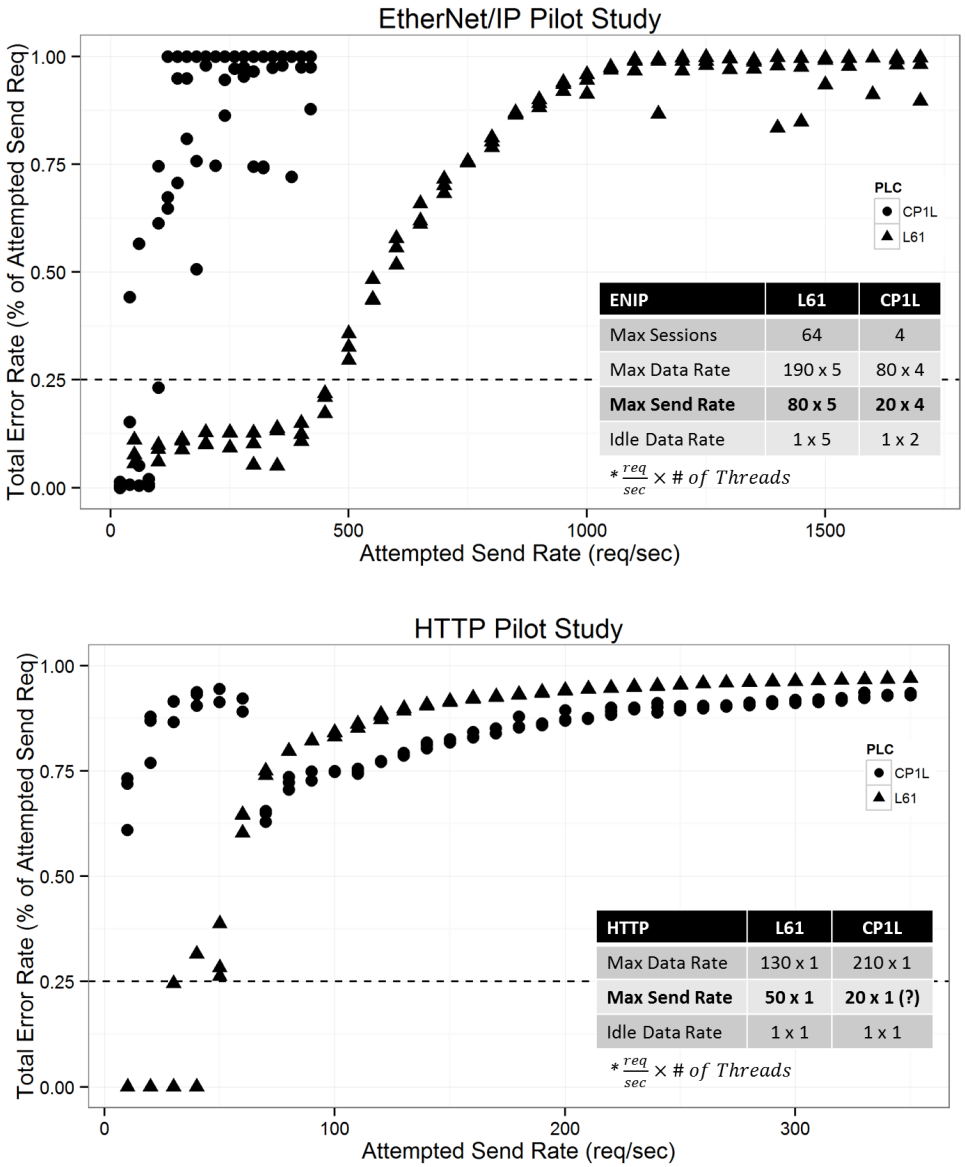


Figure 3.3. Results of pilot study 3.

Table 3.4. Reasonable data rates (requests per second) determined from pilot study 3.

	ENIP	HTTP
L61	300	50
CP1L	80	20

3.3 Improving Authenticity

An authentic network device has a unique identity (e.g., an IP address, MAC address) that distinguishes it from other network devices. Serial numbers and hostnames are examples of other types of identifiers. Some identifiers, such as the device make and model should remain constant; modifying these values would not apply in this application. The unique identity must be consistent. When an attacker connects to a device using a certain address and queries the device for its identity, the response should consistently match what the attacker expects to see, specifically with repetitive queries. Native Honeyd and PLCs both lack the capability to dynamically change these values, however, the Honeyd source code can be modified to accomplish this task. The following sections describe the modifications to Honeyd.

3.3.1 Adding Search Terms to the Honeyd Template Object.

The first modification to the Honeyd source code involves adding a linked list, called `proxy_srchitem` to the template structure in `template.h`. Each `proxy_srchitem` consists of a (char) find, (char) replace, and the respective (int) lengths of each. A conceptual illustration of the modified template data structure is shown in Figure 3.4. Storing the search terms into each template's data structure upon initialization will ensure the desired consistency. To clarify, consider a scenario where an attacker connects to a target computer using a secure shell (i.e., `ssh`) to the IP address `10.1.10.12`. If the attacker were to then query the target with a command such as `ifconfig`, they would expect the output to display the IP address `10.1.10.12`.

Repeated queries, or queries using alternate utilities (e.g., `traceroute`), should also consistently display `10.1.10.12`. If the attacker were to reconnect and attempt the same commands at a later time, the output should still display `10.1.10.12`.

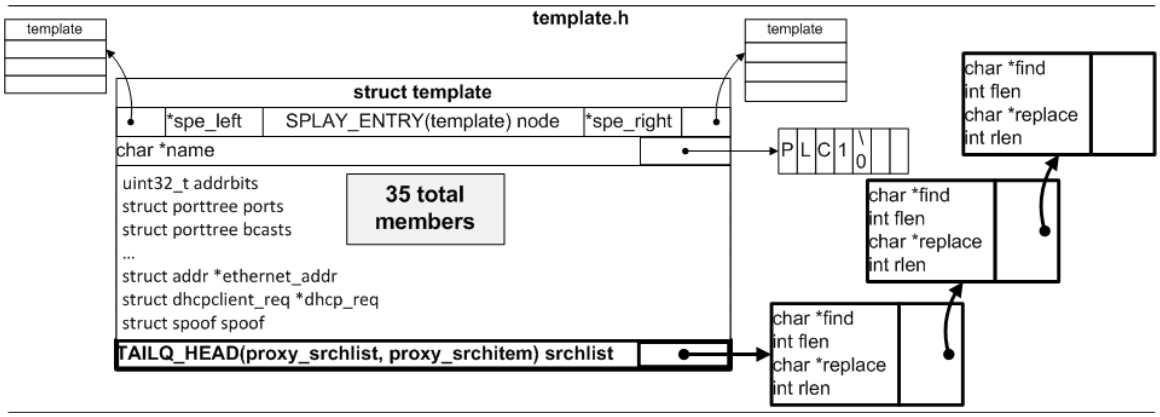


Figure 3.4. Adding the linked list of search terms to the template structure.

3.3.2 Developing the Search Terms.

The file `config.c` contains the code that implements and administers the templates, which are stored in a splay tree object (defined in `tree.h`). The second modification adds a new function `proxy_build_srchlist()` that initializes the `proxy_srchitem` list that was added to the template object.

Honeyd supports plug-in modules for enhanced functionality. The keyword `option` triggers the configuration file parser to store the line of text into a linked list of `honeyd_plugin_cfgitem` for use by plug-ins (see Figure 3.5). Immediately following the `option` keyword `option` items is a `plugin` name. The name "icsproxy" is used to identify the search terms in this modification.

In the file `plugins_config.c`, a new function called `plugins_config_find_pkg_items()` is created to extract the options denoted as "icsproxy" from the `honeyd_plugin_cfgitem` into a separate consolidated list, leaving the original `honeyd_plugin_cfgitem` list unmodified.

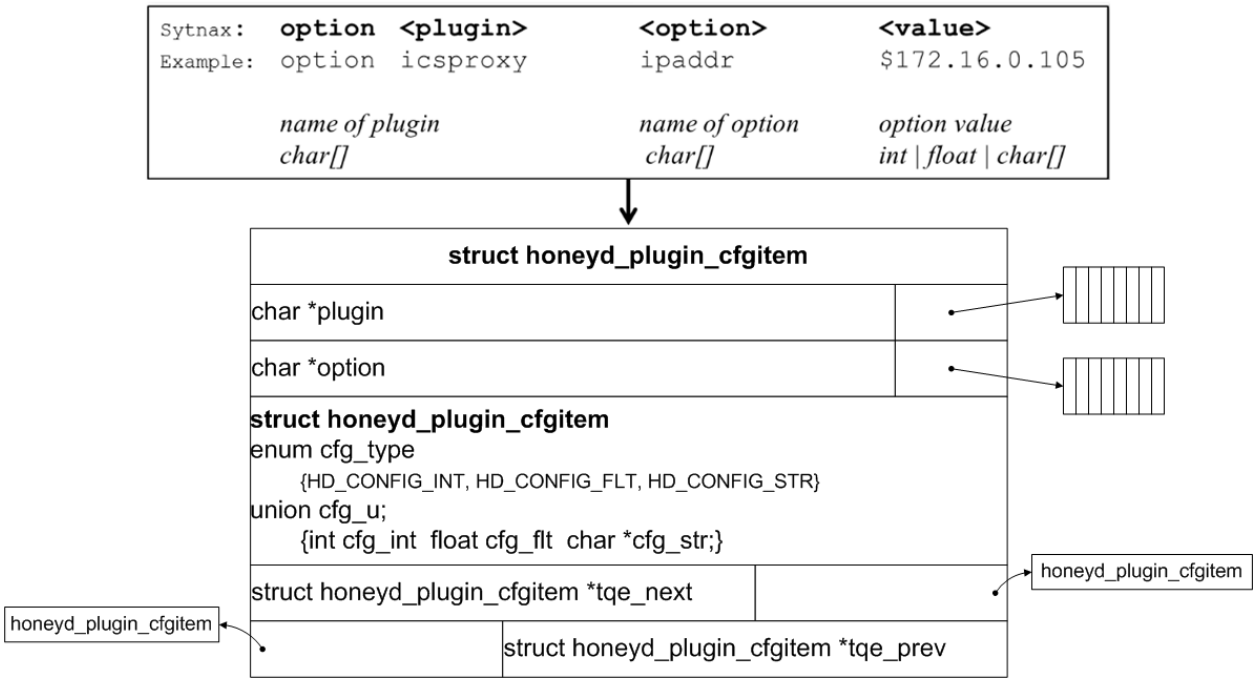


Figure 3.5. The `icsproxy` options are parsed from the configuration file (top) into a linked list (bottom) for later use.

The last step is to store the search and replace terms in each template in the template tree. The new function `proxy_update_srchlist(tmpl)` in `config.c` iterates through each search term in the icsproxy consolidated list and adds it to the template's search list. The associated replace term for the IP Address, MAC address, host name, and serial number are calculated by utility functions that were added to `config.c`. The utility functions take advantage of the data available within the template structure to generate the appropriate replace term. More protocol-specific calculated search terms can be added to the `switch()` statement in `proxy_update_srchlist` as required by specific implementations. Examples include data fields, such as a description, location, or product code used by a particular protocol. The search options can also handle static search and replace pairs, both ASCII text and raw byte patterns. Once all of the templates are populated with their search and replace terms, Honeyd completes some remaining initialization tasks, then begins listening

for connections to any of the IP addresses that are bound to templates.

3.3.3 Search and Replace on the Payload.

A new file called `icsproxy.c` contains the search and replace functions, and was inspired by code originally designed by Laird Shaw to provide a C equivalent to the PHP function `str_replace()` [28]. Shaw's code leverages the standard C `strstr` function to iteratively find all occurrences of the `find` string, and replaces them with copies of another string. However, the implementation relies on NULL terminated strings. In order to have the ability to search and replace the full range of binary values, Shaw's code is adapted to use specified string lengths instead of relying on NULL terminated ASCII strings.

In order to achieve the inline search and replace functionality, a new function `processpayload()` is inserted into `tcp_send()` and `udp_send()` in `honeyd.c`. The `processpayload()` function provides a pointer to the appropriate template, the data payload, and the payload length being handled. `processpayload()` iterates through the template's search list, calling the `replace_str()` function for each set of search and replace terms in the search list. A pointer to the modified payload is returned and the sending function continues wrapping the modified payload into a TCP/IP packet for transmission to its destination. Figure 3.6 illustrates the modifications applied to the Honeyd architecture with the added find/replace list and payload modification features.

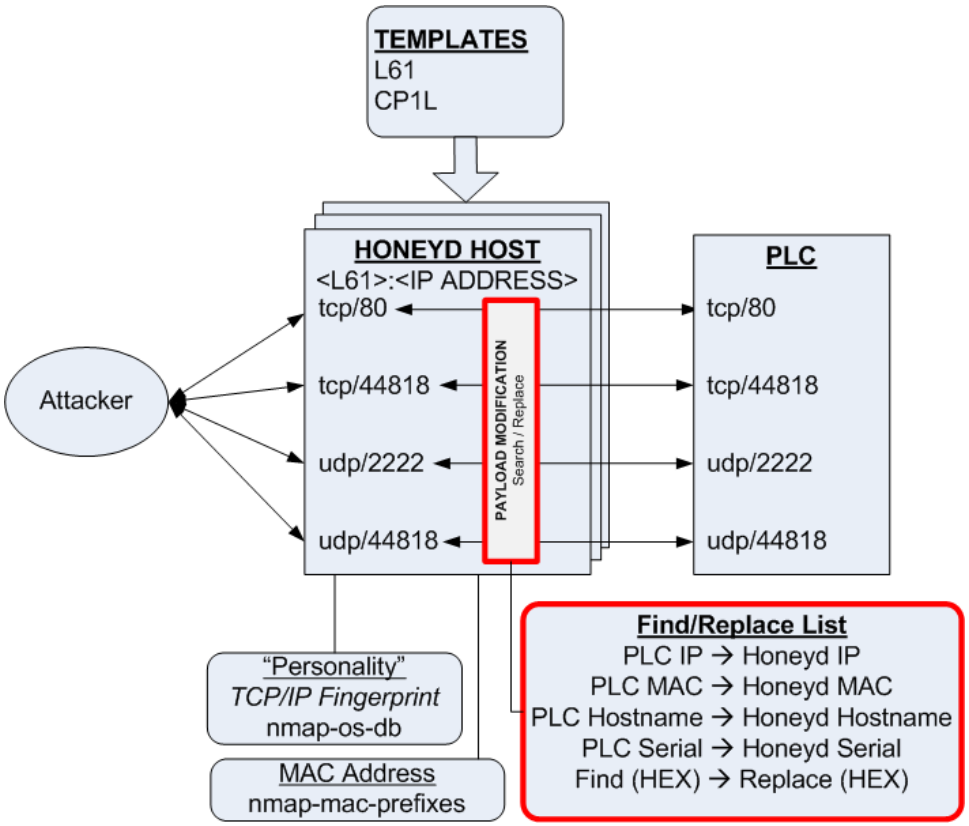


Figure 3.6. Anatomy of a Honeyd host with `icsproxy` modifications.

IV. Methodology

The evaluation approach of the Honeyd with icsproxy modifications, hereafter referred to as Honeyd+, consists of a functional test and a performance test to evaluate targetability. Section 4.1 describes the environment used to conduct the tests. Section 4.2 outlines the experimental design for the functional and performance tests. The limitations of Honeyd+ are described in Section 4.3.

4.1 Test Environment

The test environment consists of a low-cost PLC (Omron CP1L with a CP1W-EIP61 EtherNet/IP adapter) and a high-cost PLC (Allen-Bradley 1756-L61 ControlLogix, with a 1756-EWEB module). Both PLCs use HTTP and the EtherNet/IP industrial protocol. Honeyd+ is hosted on a low-cost computing platform (Raspberry Pi running Raspbian, approximately $50) and a high-cost computing platform (HP EliteBook 8570w Laptop running Ubuntu Server 14.04 LTS, approximately $1800).

Each Honeyd+ platform has two network interface cards, one is used for the public-facing IP space, where the honeypots are deployed, and the other is configured on the PLC VLAN for communication between the PLC and the Honeyd+ platform. Both Honeyd+ platforms have two configuration files, one advertising 75 hosts of the L61 template, and the other advertising 75 hosts of the CP1L template. The tests are launched from an "attacker" PC running Kali-Linux from the public-facing IP space, and the components are connected with a CISCO Catalyst 3550 switch configured with inter-VLAN routing (see Figure 4.1).

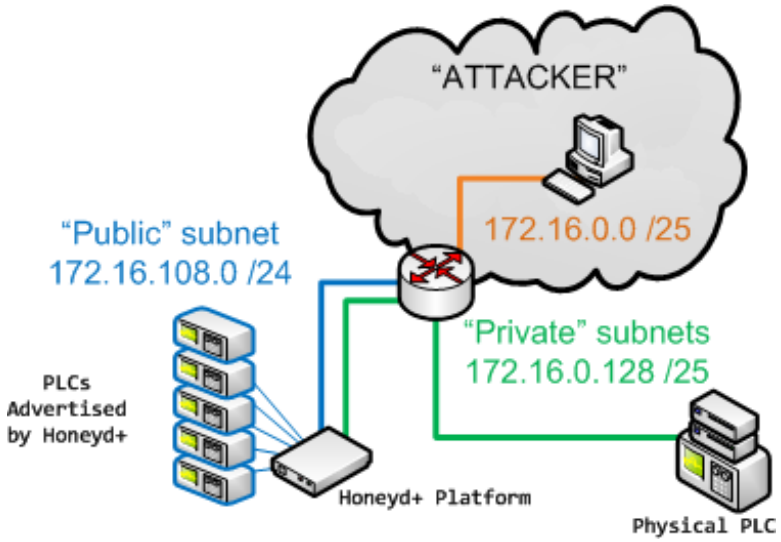


Figure 4.1. The evaluation environment.

4.2 Experimental Design

4.2.1 Functional Test.

The first test evaluates the authenticity of Honeyd+, using an Allen-Bradley L61 PLC and an OMRON CP1L PLC. Using the results from pilot testing, templates are created for both PLCs that include search terms for IP address, MAC Address, hostname, serial number, and the product name as a static search and replace pair (See Appendix C). Honeydctl is a utility function included with Honeyd that provides the operator the ability to monitor the status of Honeyd. An additional modification to Honeyd+ enabled the ability to list the search terms in each template (See Figure 4.2).

The test procedure consists of using `nmap` to conduct a port scan and operating system fingerprint of the physical PLC as well as the Honeyd+ hosts, running on the Raspberry Pi and an HP Laptop platforms. Additionally, the `wget` [14] utility downloads web pages with the HTTP protocol, and the `nmap` scripting engine (NSE) `enip-info` script tests the proxy functionality on the EtherNet/IP protocol

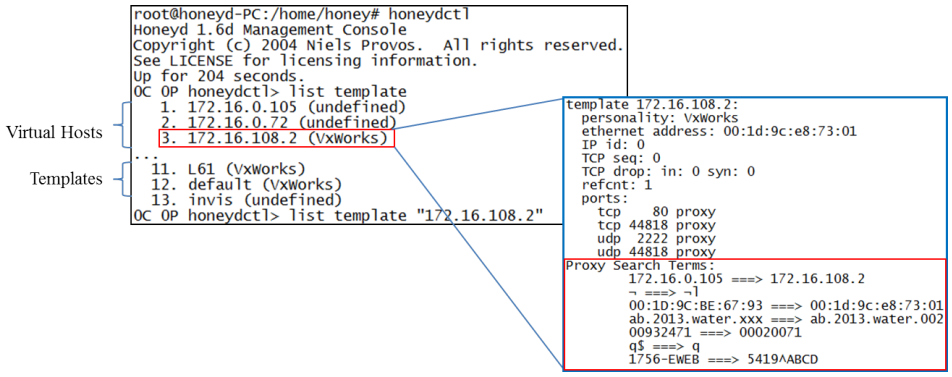


Figure 4.2. Validation of correct search terms within a template.

by sending the EtherNet/IP "`ListIdentity`" `0x0063` command [17].

The expected results are that the OS fingerprinting and port enumeration functions are preserved, and that the search terms (i.e., IP address, MAC Address, hostname, serial number, and product name) are consistently applied to give each decoy a unique identity.

The tests are repeated three times, in random order, to evaluate consistency, to account for error, and to reduce testing bias. The results are inspected, with the assistance of the `diff3` utility. The `nmap` and `wget` options that are used are shown in Tables 4.1 and 4.2.

Table 4.1. `nmap` options used for the functional test

Option	Description
`-sS`	TCP SYN Scan
`-sU`	UDP Scan
`-pT:1-1024,44818, U:1-1024,2222,44818`	Scan specified TCP/UDP ports
`-PE`	ICMP Echo probes
`-PP`	ICMP Timestamp Probes
`-A`	Enable OS detection, version detection, script scanning, and traceroute
`-script "enip-info"`	Use the NSE script "enip-info" (sends the ListIdentity command)
`-T4`	Aggressive timing
`-vv`	Very verbose output

Table 4.2. `wget` options used for the functional test

Option	Description
`-e robots=off`	Ignore the 'disallow' directives in robots.txt
`--user-agent="Mozilla/5.0 (Windows NT 6.3; rv:36.0)Gecko/20100101 Firefox/36.0"`	The L61 requires iframes, determined by checking the user agent. It redirects the default "Wget/1.13.4 (linux-gnu)" to an error page, so the –user-agent switch fools the L61 into serving up the web page to a Firefox browser.
`--max-redirect=1`	Limits the maximum number of redirections to one to keep the test within scope.

4.2.2 Performance Test.

The second test uses the Python performance testing script and the results from pilot study 3 to evaluate the performance of hosting Honeyd+ on the PC and Raspberry Pi platforms. The purpose of the test is to assess the impact of multiple simultaneous connections to the PLC.

The performance test investigates the error rates, according to Equation 3.1, when varying the factor level combinations of platform, protocol, and data rate. The response variable is the *ErrorRate*, as defined in the pilot study. *ErrorRate* is assessed according to the 25% threshold established in the pilot study, and the outcome is expected to be consistent with the observations from the pilot study. Table 4.3 illustrates the experimental design, consisting of $(1 \times 3 \times 2 \times Data\ Rate_{L61}) + (1 \times 3 \times 2 \times Data\ Rate_{CP1L})$ for a total of 48 factor-level combinations. The additional factor (*Data Rate*) is being treated as a continuous variable, but is measured at discrete intervals constrained by the fixed aggregate data rate.

The data rate is denoted as *threads* $\times$ *requests per second* (*per thread*). Each thread represents a TCP connection to the PLC from the attacker, either simultaneous connections directly to the PLC or through separate Honeyd+ hosts. There are five levels for the L61 data rate, however there are only 3 levels for the CP1L because of

the constraints discovered in the pilot study (e.g., maximum number of EtherNet/IP sessions and the limited number of HTTP requests that it could handle). The test is an incomplete factorial design, with replications because the levels of *Data Rate* are not equally represented in all factor-level combinations, due to the limitations of the factors PLC and $Protocol$.

The aggregate data rate is fixed to the values determined by pilot study 3 (Table 3.4). Each level of the variable *Data Rate* will increase the number of threads, while lowering the data rate per thread to maintain the same aggregate rate. The number of $threads \times data\ rate\ per\ thread$ is simply referred to as "rate" or "number of threads" for the remainder of this research.

Each protocol, platform, and number of threads combination is measured 50 times ($n = 50$) for a duration of 30 seconds each. To reduce bias, the PLC is rebooted prior to starting each $platform \times rate$ measurement, and the order that the measurements are performed is randomized. Each experiment is repeated three times ($r = 3$) for a total of $N = 7200$ measurements.

Table 4.3. Experimental of factors and levels.

Factor	**Experiment 1**	
PLC (categorical)	CP1L	
Platform (categorical)	Baseline-CP1L PC-CP1L Pi-CP1L	
Protocol (categorical)	EtherNet/IP	HTML
Data Rate* (interval)	1×80 2×40 4×20	1×20 2×10 4×5
Aggregate Data Rate	80	20

Factor	**Experiment 2**	
PLC (categorical)	L61	
Platform (categorical)	Baseline-L61 PC-L61 Pi-L61	
Protocol (categorical)	EtherNet/IP	HTML
Data Rate* (interval)	4×75 5×60 12×25 10×30 20×15	1×50 2×25 5×10 10×5 25×2
Aggregate Data Rate	300	50

*fixed data rate, in requests per second denoted as $Threads \times Rate$

4.3 Limitations

There are some limitations to consider prior to implementing Honeyd+. First, the search and replace functionality was tested on protocols that use length-based error checking, such as EtherNet/IP. Protocols that use encrypted or compressed communications are not supported. Protocol-specific algorithms, such as error checking, encryption, and compression could potentially be incorporated into the Honeyd+ source code, but is not in the scope of this research.

Provos demonstrated that a traditional implementation of Honeyd could support 2,000 TCP requests per second among 65,536 hosts [24]. The pilot study showed that PLCs can support significantly less connections, depending on the manufacturer, protocol, and model. Additionally, since multiple Honeyd+ hosts communicate with a single PLC, any modifications or degradation of the PLC will immediately reflect on all of the Honeyd+ hosts.

V. Results and Analysis

The functional and performance tests completed successfully without complications, or discernible errors. The functional test demonstrated the authenticity of Honeyd+ in all of the tested cases. The performance test focused on the targetability characteristic of Honeyd+ and was successful in the lower data rates for the EtherNet/IP protocol, however error rates at the higher data rates were not optimal. The HTTP protocol performance was inconclusive. Section 5.1 discusses the results of the functional test, and Section 5.2 discusses the results of the performance test.

5.1 Functional Test

Advertising a subnet of 75 Honeyd+ hosts with the five search terms in Figure 5.1 (e.g., ipaddr, ethaddr, hostname, serial, hex) produced similar results as in Figure 3.1. Deeper examination of the `nmap` port enumeration and OS fingerprint, inspection of the web pages, and the response from the `ListIdentity` command demonstrated accurate and consistent application of the search terms in all cases and all repetitions, denoted by a ✓in Table 5.1. Figures 5.2, 5.3, and 5.4 highlight the changes on one of the Honeyd+ hosts compared to the physical PLC on each of the tools tested.

Table 5.1. Results of functional test

Device	Platform	Protocol	nmap enum	nmap port scan +OS fingerprint	wget	nmap NSE enip-info
CP1L	Baseline	EtherNet/IP	✓	✓	✓	✓
		HTML	✓	✓	✓	✓
	PC	EtherNet/IP	✓	✓	✓	✓
		HTML	✓	✓	✓	✓
	Pi	EtherNet/IP	✓	✓	✓	✓
		HTML	✓	✓	✓	✓
L61	Baseline	EtherNet/IP	✓	✓	✓	✓
		HTML	✓	✓	✓	✓
	PC	EtherNet/IP	✓	✓	✓	✓
		HTML	✓	✓	✓	✓
	Pi	EtherNet/IP	✓	✓	✓	✓
		HTML	✓	✓	✓	✓

(✓= *Accurate and consistent application of all search terms*)

```
## L61 Search Terms
option icsproxy ipaddr $172.16.0.105

option icsproxy ethaddr $00-1D-9C-BE-67-93

option icsproxy hostname $eWeb_xxx

option icsproxy serial $00932471

option icsproxy $(hex)313735362d45574542 $(hex)3534313900393324716768
# option icsproxy      1 7 5 6 - E W E B        5 4 1 9\x 9 3 $ q C D

## CP1L Search Terms
option icsproxy ipaddr $172.16.0.106

option icsproxy ethaddr $00-1D-4B-F0-22-66

option icsproxy hostname $CP1W-EIP61

option icsproxy serial $4bf02266

option icsproxy $(hex)435031572d4549503631 $(hex)524a3558214e51413932
# option icsproxy      C P 1 W - E I P 6 1        R J 5 X ! N Q A 9 2
```

Figure 5.1. Search terms used in the authenticity test.

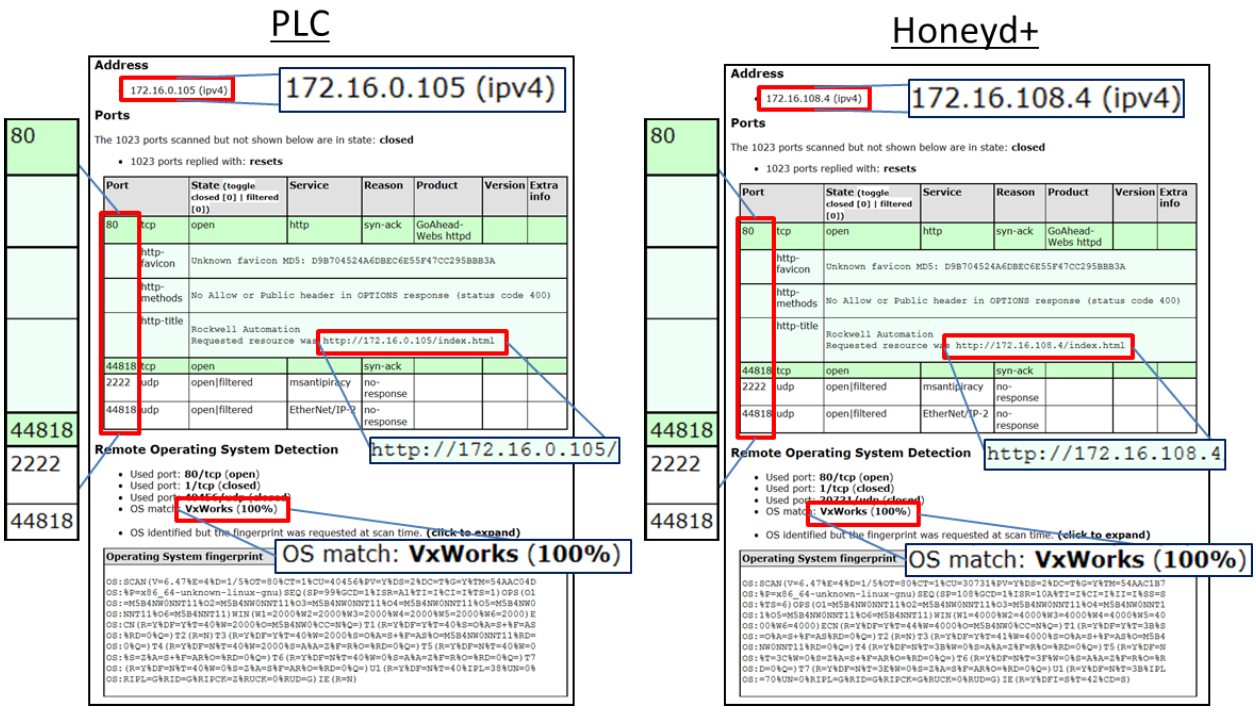


Figure 5.2. `nmap` **port enumeration and fingerprint: The actual PLC (left) and the Honeyd+ host response (right).**

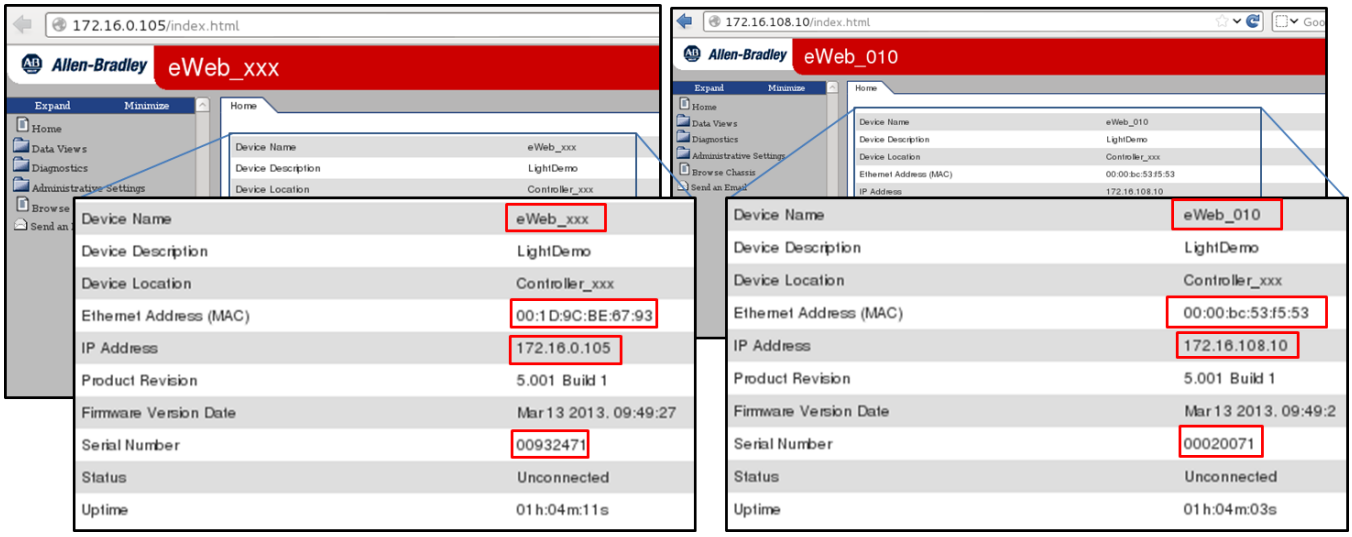


Figure 5.3. Web page response: the actual PLC web page (left) and the Honeyd+ host web page(right).

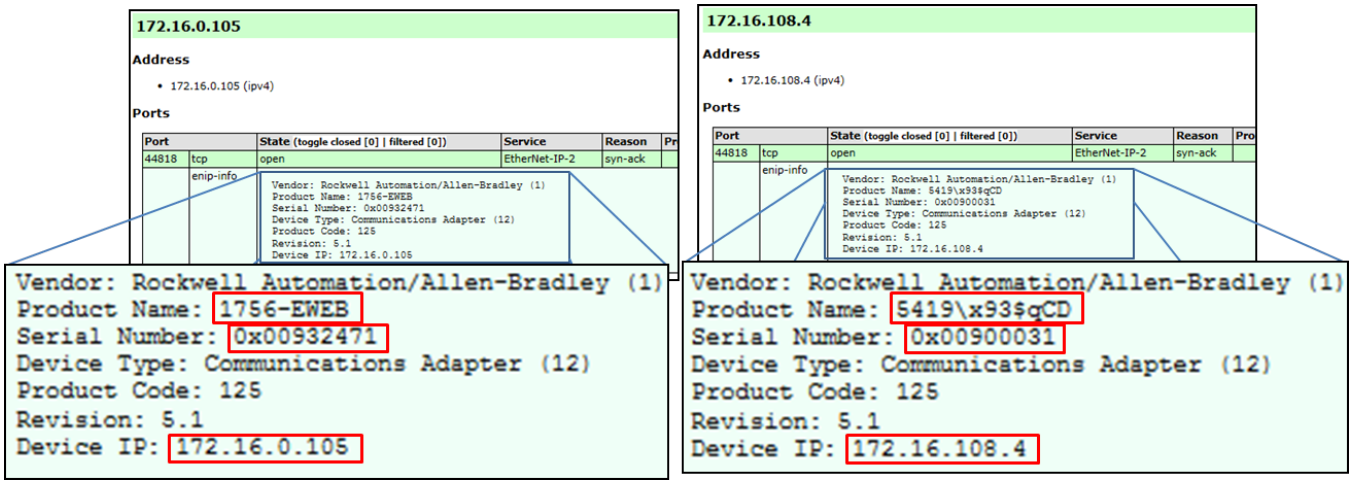


Figure 5.4. The `ListIdentity (0x0063)` command via `nmap` NSE script: The actual PLC (left) and the Honeyd+ host response (right).

5.2 Performance Test

Overall, the EtherNet/IP protocol had an increased mean error rate at each level across all of the platforms as the number of connections increased, despite the constant aggregate data rate. For the HTTP protocol, the average mean error rate on the L61 was higher–yet nearly identical–on the Raspberry Pi and PC platform than the Baseline. The mean error rate for HTTP protocol on the CP1L was approximately 88% across all platforms, which is consistent with observations from the pilot test. A summary of the results is shown Tables 5.2 and 5.3 and are plotted on interaction plots in Figures 5.6-5.8; verbose data tables are in Appendix D. With the exception of the CP1L-Baseline platform, the variances were all <1%.

Table 5.2. Summarized mean error rates for the CP1L

ENIP (80)	Physical PLC no Honeyd	Honeyd on PC	Honeyd on Raspberry Pi
1x80	0.00%	0.00%	0.00%
2x40	3.45%	0.18%	2.40%
4x20	11.04%	28.36%	29.51%
Total (avg)	**4.83%**	**9.51%**	**10.64%**

HTTP (20)	Physical PLC no Honeyd	Honeyd on a PC	Honeyd on Raspberry Pi
1x20	87.66%	88.66%	88.73%
2x10	85.77%	88.87%	89.14%
4x5	86.77%	88.68%	88.93%
Total (avg)	**86.74%**	**88.74%**	**88.93%**

Table 5.3. Summarized mean error rates for the L61

ENIP (300)	Physical PLC no Honeyd	Honeyd on PC	Honeyd on Raspberry Pi
4x75	9.3%	11.4%	7.9%
5x60	14.4%	16.4%	12.7%
12x25	29.5%	30.1%	27.4%
10x30	35.1%	36.2%	33.4%
20x15	58.7%	63.9%	60.1%
Total (avg)	**29.4%**	**31.6%**	**28.3%**

HTTP (50)	Physical PLC no Honeyd	Honeyd on a PC	Honeyd on Raspberry Pi
1x50	0.7%	32.4%	32.1%
2x25	0.1%	32.5%	32.2%
5x10	14.1%	32.3%	31.9%
10x5	18.0%	32.3%	31.9%
25x2	54.1%	58.5%	57.4%
Total (avg)	**17.4%**	**37.6%**	**37.1%**

5.2.1 Evaluation.

Evaluating the error rates of on the CP1L with the HTTP protocol (Figure 5.5) against the 25% threshold showed a consistent error rate of approximately 88% on all three platforms. The pilot study noted an approximate 60% error rate for the

CP1L/HTTP combination; the outcome is higher than the 25% threshold is expected.

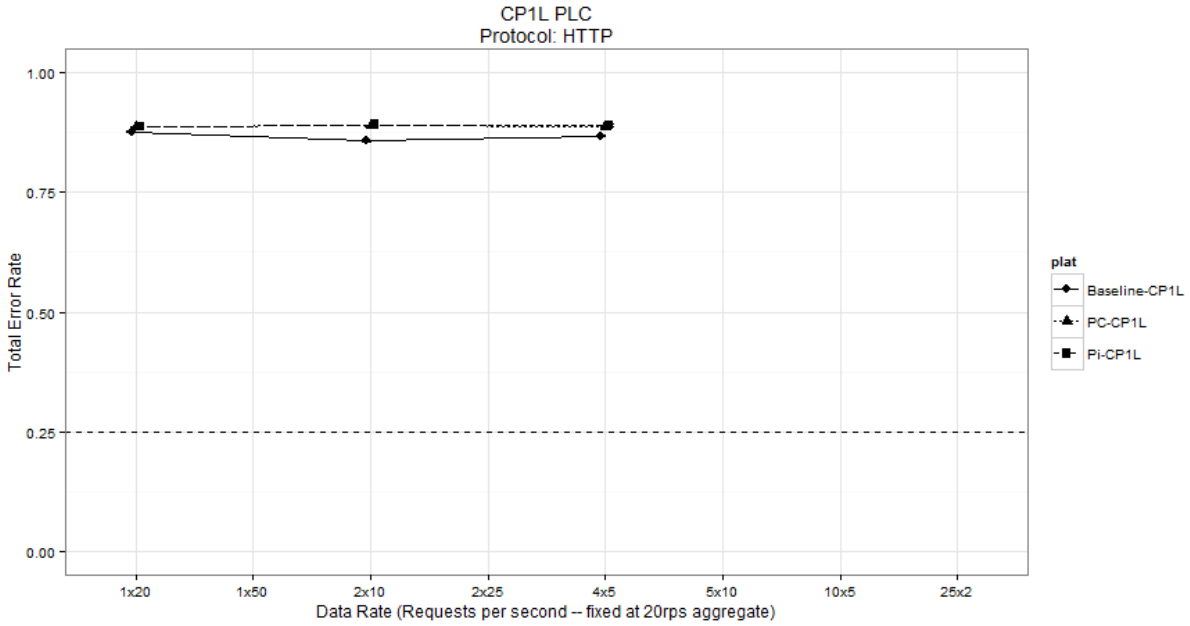


Figure 5.5. Interaction plots of error rate versus number of threads for the CP1L PLC and HTTP protocol.

The HTTP protocol on the L61 (Figure 5.6) was inconsistent with the expectations from the pilot study. The error rate on the baseline platform increased, though remaining under the 25% threshold, until the highest rate (25×2). However, the error rates for the Honeyd+ platforms were consistently 32% until reaching the 25×2 rate. The interaction plot indicates that there was no effect between either of the Honeyd+ platforms and the physical PLC for the 1×50 through the 10×5 rates.

The EtherNet/IP Protocol on the CP1L (Figure 5.7) performed slightly better than the pilot study, with the 1×80 and the 2×40 rates having error rates less then 5%, however, the error rates for both of the Honeyd+ platforms exceeded the 25% threshold when the rate increased to 4×20. At the 4×20 rate, the CP1L PLC still performed well under the threshold, with an 11% error rate. An observation for further study is that the variance on the CP1L-Baseline seems to expand as the rate

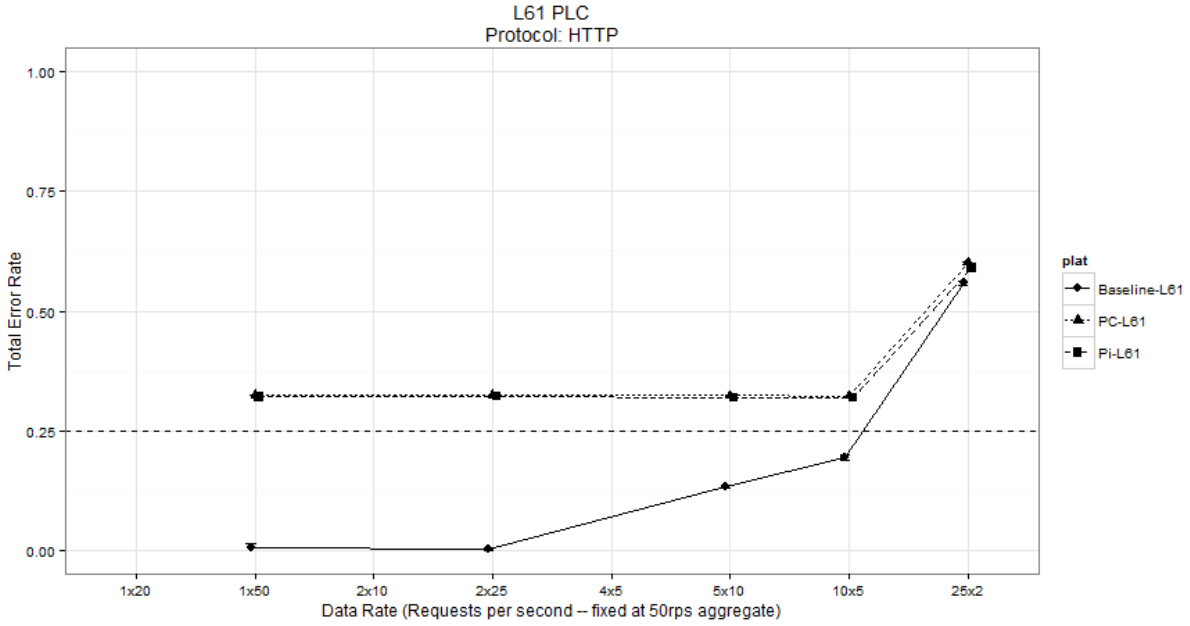


Figure 5.6. Interaction plots of error rate versus number of threads for the L61 PLC and HTTP protocol.

increases, which does not occur with the CP1L Honeyd+ platforms or on the L61 platforms.

The EtherNet/IP Protocol on the L61 (Figure 5.8) showed a consistent increase in error rate on all three platforms as the rate increased. Most noteworthy is that the Raspberry Pi platform outperformed the baseline and the PC platforms on this test. While this thesis offers no explanation for this finding, it should be further studied.

Overall, as the rate increased, all four PLC and rate combinations performed less optimally than expected. The conclusion is that a PLC can support approximately 5 simultaneous connections—10 in the best case—at the 25% error rate threshold. The results of the performance evaluation are summarized in Table 5.4.

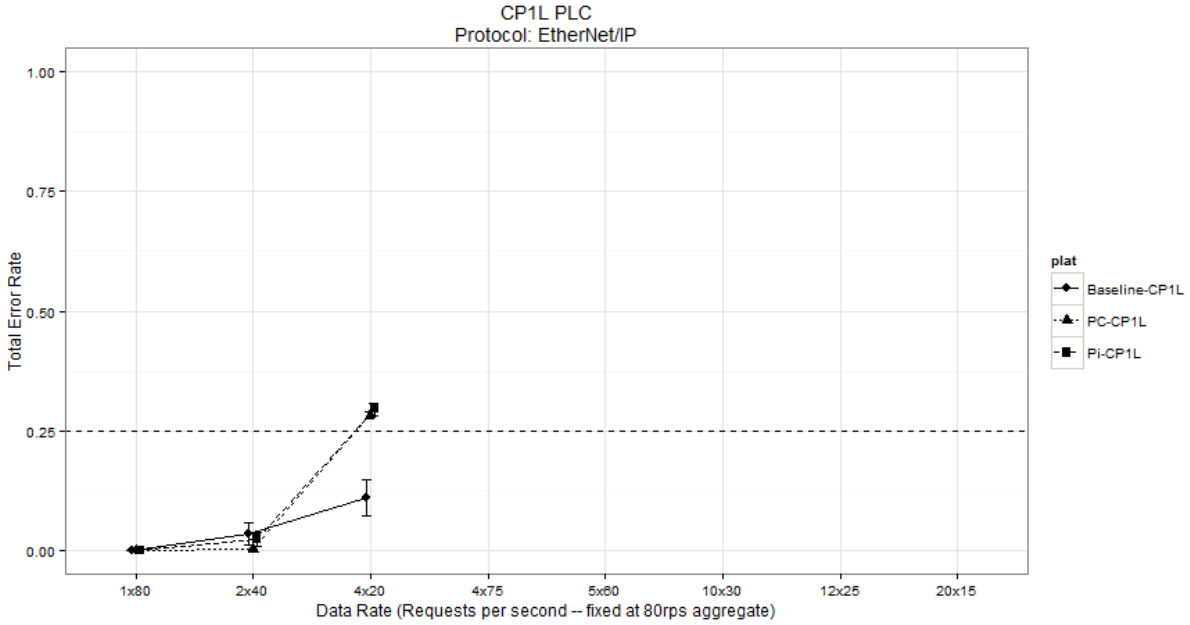


Figure 5.7. Interaction plots of error rate versus number of threads for the CP1L PLC and EtherNet/IP protocol.

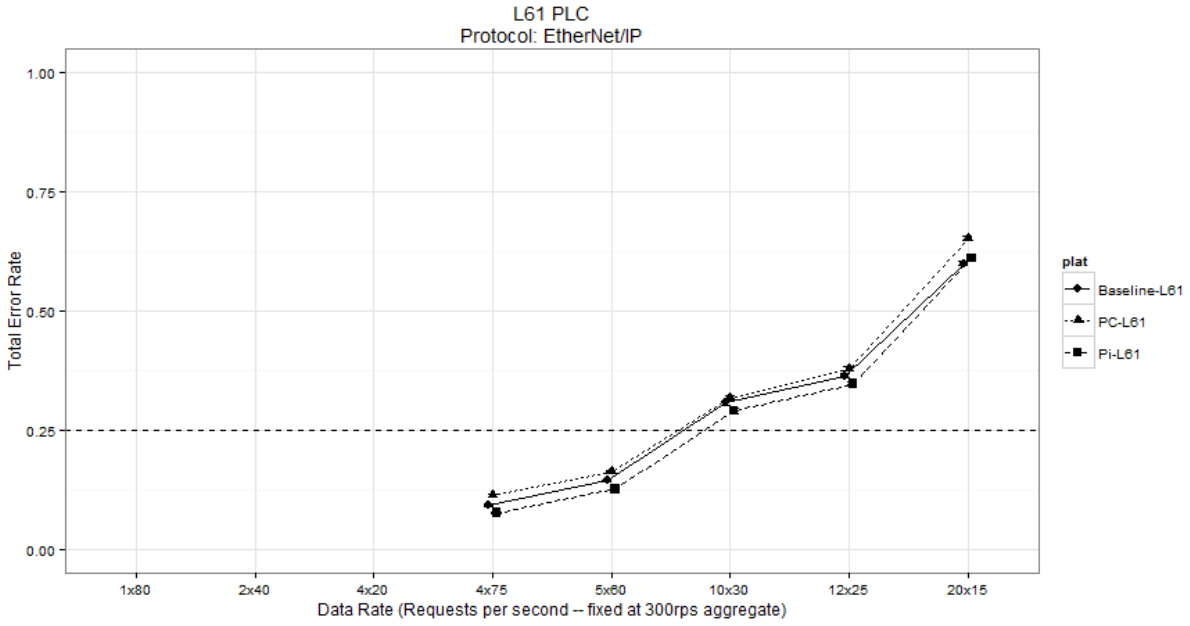


Figure 5.8. Interaction plots of error rate versus number of threads for the L61 PLC and EtherNet/IP protocol.

Table 5.4. Results of performance test.

Protocol	Platform	1x80	2x40	4x20
EtherNet / IP	CP1L-Baseline	✓	✓	✓
	CP1L-PC	✓	✓	**X**
	CP1L-Pi	✓	✓	**X**
		1x20	2x10	4x5
HTTP	CP1L-Baseline	**X**	**X**	**X**
	CP1L-PC	**X**	**X**	**X**
	CP1L-Pi	**X**	**X**	**X**

Protocol	Platform	4x75	5x60	10x30	12x25	20x15
EtherNet / IP	L61-Baseline	✓	✓	**X**	**X**	**X**
	L61-PC	✓	✓	**X**	**X**	**X**
	L61-Pi	✓	✓	**X**	**X**	**X**
		1x50	2x25	5x10	10x5	25x2
HTTP	L61-Baseline	✓	✓	✓	✓	**X**
	L61-PC	**X**	**X**	**X**	**X**	**X**
	L61-Pi	**X**	**X**	**X**	**X**	**X**

✓= *Error Rate* <25% **X** = *Error Rate*>25%

VI. Conclusion

This chapter summarizes the overall conclusions of the research. Section 6.1 presents conclusions based on the results and analysis from Chapter 5. Section 6.2 discusses the significance of this research. Section 6.3 discusses recommendations and approaches for future research.

6.1 Research Conclusions

The tests conducted in this research demonstrated that Honeyd+ could represent 75 authentic PLCs on both a laptop PC and a raspberry Pi, each containing 5 search terms. Such a scale demonstrates significant cost savings over purchasing physical equipment, and demonstrates targetability and authenticity. The results of the performance test of Honeyd+ identified potential targetability limitations, though it is still a cost saving alternative to deploying multiple physical honeypots.

6.1.1 Functional Test.

The functional test demonstrated that Honeyd+ could accurately identify five search terms (IP Address, MAC Address, Host name, Serial Number, and Product Name) and accurately replace them with appropriate values. The resulting decoys have unique and consistent identities from the attacker's perspective, resulting in a high degree of authenticity.

6.1.2 Performance Test.

The PLCs supported less than ten simultaneous connections at its maximum effective data rate, as defined and determined by a pilot study. Further study with different devices, data rates, and protocols may discover a testing design with more

optimal results. A Honeyd+ host does not communicate with the PLC unless an attacker communicates with the port that is proxied to the PLC (i.e., Honeyd+ bears a majority of the load for a port scan and OS fingerprint). Furthermore, performance analysis revealed that the low-cost Raspberry Pi performed about the same–or better– as a high cost laptop PC, making the Raspberry Pi the preferrable option to a low-cost honeypot solution.

6.1.3 Research Hypothesis.

The results of the functional test confirms the hypothesis that Honeyd+ presents an accurate representation of many PLCs. Figures 5.6-5.8 illustrates different performance behaviors, based on protocol and device manufacturer. The hypotheses that the performance limitations of a PLC is the limiting factor of performance in the honeypot implementation is confirmed, though the practical impact is inconclusive. Further study is required to quantify the impact to a production implementation. An interesting contradiction to the initial research hypothesis is that the Rasperry Pi platform performed equal to or better than the PC platform in all four tested cases.

6.2 Significance of Research

The authenticity of a physical PLC is ideal however, a single PLC can cost over $5,000. The cost of implementing multiple highly authentic physical PLCs as honeypots quickly becomes prohibitive, which is a constraint on targetability. In contrast, emulators can be replicated on various low-cost hardware platforms to gain a high degree of targetability, but their authenticity is constrained to the features and behaviors contained in the programming code of the emulator. Lack of a standard ICS virtualization environment further restricts the options to create ICS honeypots that have a high degree of of authenticity and targetability with minumum risks and

costs. This research demonstrated a technique that achieves both authenticity and targetability, with a cost savings ranging from 5 honeypots (worst case) to 75 (best case) decoy PLCs for the cost of a single PLC and a Raspberry Pi.

6.3 Future Work

6.3.1 Test Different PLCs Manufacturer Protocols.

This research tested the function and performance for two protocols each on two different PLCs. Examination of additional devices and protocols using the methodology in this research can create a more comprehensive database of PLC templates and may discover devices that perform better under a variety of circumstances.

6.3.2 Develop Additional Search Term Capabilities.

Honeyd+ demonstrated five basic search terms that are common to most networked devices. Additional functions can be inserted into the source code to support protocol specific algorithms (e.g., error checking, encryption, and compression) or other desired data points for a specific implementation.

6.3.3 Compensate for the Limitations of the PLC.

Additional modifications or alternate techniques can compensate for the limitations of the PLC. For example, substituting a script or separate web server for the HTTP protocol. Another example would incorporating some form of caching functionality or a stateful memory program to reduce the burden of repetitive communication with the PLC. Finally, it is possible to configure templates to proxy to more than one PLC for a load balancing enhancement, although this technique could dilute the desired cost benefits.

6.3.4 Deploy on a Network.

Honeyd+ clearly has the potential application as the research honeypot. Honeyd+ inherited the additional capabilities of native Honeyd, such as simulating the routing of multiple network segments. Other ICS components, such as HMI, sensors, or actuators could be added to create a more comprehensive honeynet.

Bodenheim recommended deployment on an Amazon EC2 platform. Honeyd+ makes it feasible to pursue this research effort, assuming the additional cost of hosting and IP space is acceptable.

Another potential application for Honeyd+ is a honeynet of decoys on a production network. A Honeyd+ template on Raspberry Pi computers installed at multiple remote sites could cloak the real assets while using one centrally located PLC (see Figure 6.1). Such an application would increase an attacker's chance of interacting with a honeypot during the reconnaissance and enumeration stages, giving asset owners an opportunity to respond to the intrusion.

Logging is an important consideration when deploying such a honeynet. Specific logging capabilities were not considered in this research, however, Honeyd+ includes the simple logging capability from native Honeyd that catalogs attempted connections by source IP, source port, and operating system fingerprint. More verbose monitoring on the centrally located PLC with available open source tools, such as Wireshark, would enhance the logging with a full packet capture ability. A management tool, such as Honeywall [8], can centralize the management of multiple honeypots into a web-based interface with report generation capabilities.

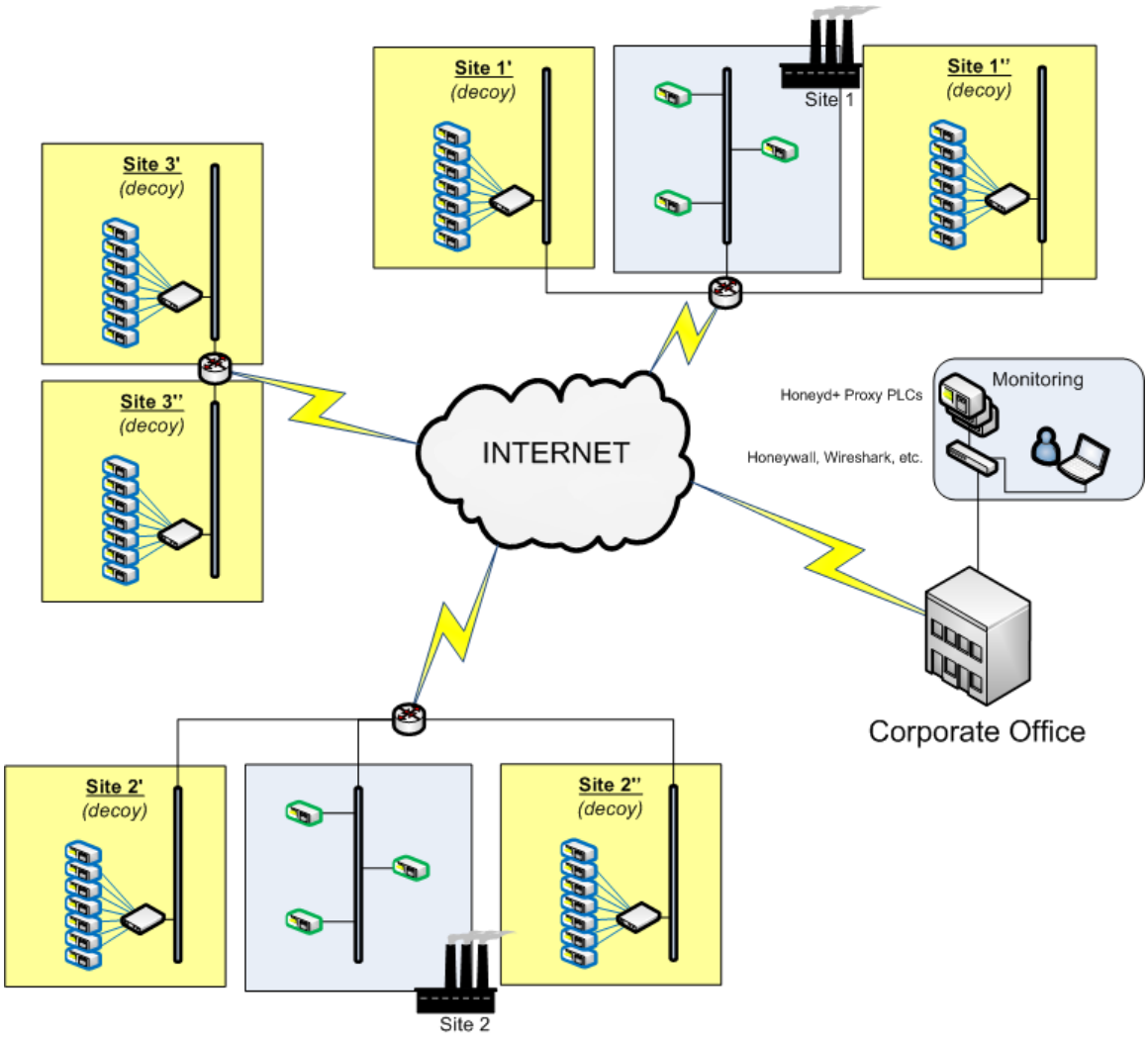


Figure 6.1. Potential implementation of Honeyd+ as a production honeynet.

Appendix A. Nmap Results from Pilot Study 1

Verbose nmap fingerprint and enumeration of the CP1L PLC

```
# Nmap 6.40 scan initiated Mon Dec  1 19:59:26 2014 as: nmap -e eth0 -oA plc_scan_106 -sC -sS -sU -T4 -A -vv
      -PE -PP -p* --script discovery --stats-every 30s 172.16.0.106
Pre-scan script results:
| broadcast-eigrp-discovery:
|_ ERROR: Couldn't get an A.S value.
| broadcast-igmp-discovery:
|   172.16.0.200
|     Interface: eth0
|     Version: 2
|     Group: 224.0.0.252
|     Description: Link-local Multicast Name Resolution (rfc4795)
|   172.16.0.202
|     Interface: eth0
|     Version: 2
|     Group: 224.0.0.251
|     Description: mDNS
|   172.16.0.200
|     Interface: eth0
|     Version: 2
|     Group: 239.255.255.250
|     Description: Organization-Local Scope (rfc2365)
|_  Use the newtargets script-arg to add the results as targets
| broadcast-ping:
|   IP: 172.16.0.108  MAC: 00:1d:9c:a3:dc:8d
|   IP: 172.16.0.106  MAC: 00:1d:4b:f0:22:66
|_  Use --script-args=newtargets to add the results as targets
| http-icloud-findmyiphone:
|_  ERROR: No username or password was supplied
| http-icloud-sendmsg:
|_  ERROR: No username or password was supplied
| targets-asn:
|_  targets-asn.asn is a mandatory parameter
| targets-sniffer: Sniffed 5 address(es).
| 224.0.0.1
| 172.16.0.108
| 172.16.0.106
| 172.16.0.200
|_239.255.255.250
Increasing send delay for 172.16.0.106 from 0 to 5 due to 71 out of 177 dropped probes since last increase.
Nmap scan report for 172.16.0.106
Host is up (0.0021s latency).
Scanned at 2014-12-01 19:59:38 EST for 737s
Not shown: 19850 closed ports
PORT      STATE         SERVICE        VERSION
80/tcp    open          http?
|_http-chrono: Request times for /; avg: 16000.65ms; min: 16000.52ms; max: 16000.83ms
|_http-comments-displayer: Couldn't find any comments.
|_http-default-accounts: [ERROR] HTTP request table is empty. This should not happen since we at least made
     one request.
|_http-drupal-modules: ERROR: Script execution failed (use -d to debug)
```

```
|_http-google-malware: [ERROR] No API key found. Update the variable APIKEY in http-google-malware or set it
      in the argument http-google-malware.api
| http-grep:
|_  ERROR: Argument http-grep.match was not set
| http-headers:
|_  (Request type: GET)
| http-sitemap-generator:
|   Directory structure:
|   Longest directory structure:
|     Depth: 0
|     Dir: /
|   Total files found (by extension):
|_
|_http-traceroute: ERROR: Script execution failed (use -d to debug)
| http-vhosts:
|_28 names had status ERROR
|_http-waf-detect: [ERROR] Initial HTTP request failed
|_http-wordpress-plugins: ERROR: Script execution failed (use -d to debug)
9999/tcp  open          abyss?
44818/tcp open          EtherNet/IP-2?
69/udp    open          tftp?
2222/udp  open|filtered msantipiracy
30718/udp open|filtered unknown
44818/udp open|filtered EtherNet/IP-2
3 services unrecognized despite returning data. If you know the service/version, please submit the following
      fingerprints at http://www.insecure.org/cgi-bin/servicefp-submit.cgi :
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)==============
SF-Port80-TCP:V=6.40%I=7%D=12/1%Time=547D0F42%P=x86_64-unknown-linux-gnu%r
SF:(GetRequest,38F,"HTTP/1\.1\x20200\r\nContent-type:\x20text/html\r\n\r\n
SF:<HTML><HEAD><TITLE>Omron\x20CP1W-EIP61\x20Configuration</TITLE></HEAD><
SF:BODY><H2>CP1W-EIP61\x20Configuration</H2><FORM\x20action=\"postcfg\.cgi
SF:\"\x20method=\"post\">\n<TABLE>\n<TR>\n<TD>IP\x20Address:</TD>\n<TD><in
SF:put\x20type=text\x20name=\"ipaddr\"\x20size=15\x20maxlength=15\x20value
SF:=172\.16\.0\.106></TD>\n</TR>\n<TR>\n<TD>Subnet\x20Mask:</TD>\n<TD><inp
SF:ut\x20type=text\x20name=\"subnet\"\x20size=15\x20maxlength=15\x20value=
SF:255\.255\.255\.0></TD>\n</TR>\n<TR>\n<TD>Gateway:</TD>\n<TD><input\x20t
SF:ype=text\x20name=\"gateway\"\x20size=15\x20maxlength=15\x20value=0\.0\.
SF:0\.0></TD>\n</TR>\n<TR>\n<TD>FINS\x20UDP\x20Port:</TD>\n<TD><input\x20t
SF:ype=text\x20name=\"finsudp\"\x20size=15\x20maxlength=5\x20value=9600></
SF:TD>\n</TR>\n</TABLE>\n<P>Behavior\x20on\x20Loss\x20of\x20I/O\x20Connect
SF:ion:<BR>\n<input\x20type=radio\x20name=\"lost\"\x20value=\"same\"\x20>C
SF:IO\x20Outputs\x20Stay\x20the\x20Same<BR>\n<input\x20type=radio\x20name=
SF:\"lost\"\x20value=\"zero\"\x20checked>CIO\x20Outputs\x20Set\x20to\x20Ze
SF:ro<BR>\n<BR><INPUT\x20type=\"submit\"\x20value=\"\x20Apply\x20Setting\x
SF:20\"></FORM></B");
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)==============
SF-Port9999-TCP:V=6.40%I=7%D=12/1%Time=547D0F42%P=x86_64-unknown-linux-gnu
SF:%r(NULL,8F,"\xff\xfb\x03\xff\xfd\x18\r\n\r\n\0MAC\x20addre\xff\xfb\x03s
SF:s\x2000:1D:4B:F0:22:66\r\nSoftware\x20version\x201\.01\x20\(Oct\x20\x20
SF:5\x202011\)\x20CP1W-EIP61\r\n\r\n\0Setup\x20Menu\r\n\r\nPress\x20Enter\
SF:x20to\x20go\x20into\x20Setup\x20Mode")%r(GetRequest,92,"\xff\xfb\x03\xf
SF:f\xfd\x18\r\n\r\n\0MAC\x20addre\xff\xfb\x03ss\x2000:1D:4B:F0:22:66\r\nS
SF:oftware\x20version\x201\.01\x20\(Oct\x20\x205\x202011\)\x20CP1W-EIP61\r
SF:\n\r\n\0Setup\x20Menu\r\n\r\nPress\x20Enter\x20to\x20go\x20into\x20Setu
```

```
SF:p\x20Mode\?!\?")%r(HTTPOptions,2C2,"\xff\xfb\x03\xff\xfd\x18\r\n\r\n\0M
SF:AC\x20addre\xff\xfb\x03ss\x2000:1D:4B:F0:22:66\r\nSoftware\x20version\x
SF:201\.01\x20\(Oct\x20\x205\x202011\)\x20CP1W-EIP61\r\n\r\n\0Setup\x20Men
SF:u\r\n\r\nPress\x20Enter\x20to\x20go\x20into\x20Setup\x20Mode\r\n\n\0\*\
SF:*\*\x20basic\x20parameters\x20\r\n\r\0Hardware:\x20Ethernet\x20TPI\r\n\
SF:0IP\x20addr\x20172\.16\.0\.106\r\n\0Netmask\x20255\.255\.255\.0\r\n\0No
SF:\x20Gateway\x20Set\r\n\r\n\0\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\x20Chann
SF:el\x201\x20\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n\r\0Serial\x20Settings
SF:\x20\x20:\x20115200\x20baud\x20\x20RS232,8,N,1\x20\x20Flow:None\r\n\r\n
SF:\r\n\r\n\r\0Change\x20Setup:\r\n\0\x20\x200\x20Server\x20configuration\
SF:r\n\r\0\x20\x206\x20debug\r\n\r\0\x20\x207\x20factory\x20defaults\r\n\r
SF:\0\x20\x208\x20exit\x20without\x20save\r\n\r\0\x20\x209\x20save\x20and\
SF:x20exit\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20
SF:Your\x20choice\x20\?\x20\x07\x07\x07\x07\x07\x07\x07\x07\x07\x07\x07\x0
SF:7\x07\x07\x071\r\n\r\n\r\n\r\0Change\x20Setup:\r\n\0\x20\x200\x20Server
SF:\x20configuration\r\n\r\0\x20\x206\x20debug\r\n\r\0\x20\x207\x20factory
SF:\x20defaults\r\n\r\0\x20\x208\x20exit\x20without\x20save\r\n\r\0\x20\x2
SF:09\x20save\x20and\x20exit\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x
SF:20\x20\x20\x20\x20Your\x20choice\x20\?\x20")%r(FourOhFourRequest,423,"\
SF:xff\xfb\x03\xff\xfd\x18\r\n\r\n\0MAC\x20addre\xff\xfb\x03ss\x2000:1D:4B
SF::F0:22:66\r\nSoftware\x20version\x201\.01\x20\(Oct\x20\x205\x202011\)\x
SF:20CP1W-EIP61\r\n\r\n\0Setup\x20Menu\r\n\r\nPress\x20Enter\x20to\x20go\x
SF:20into\x20Setup\x20Mode\r\n\n\0\*\*\*\x20basic\x20parameters\x20\r\n\r\
SF:0Hardware:\x20Ethernet\x20TPI\r\n\0IP\x20addr\x20172\.16\.0\.106\r\n\0N
SF:etmask\x20255\.255\.255\.0\r\n\0No\x20Gateway\x20Set\r\n\r\n\0\*\*\*\*\
SF:*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\x20Channel\x201\x20\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
SF:\*\*\*\*\*\r\n\r\0Serial\x20Settings\x20\x20:\x20115200\x20baud\x20\x20RS
SF:232,8,N,1\x20\x20Flow:None\r\n\r\n\r\n\r\n\r\0Change\x20Setup:\r\n\0\x2
SF:0\x200\x20Server\x20configuration\r\n\r\0\x20\x206\x20debug\r\n\r\0\x20
SF:\x207\x20factory\x20defaults\r\n\r\0\x20\x208\x20exit\x20without\x20sav
SF:e\r\n\r\0\x20\x209\x20save\x20and\x20exit\x20\x20\x20\x20\x20\x20\x20\x
SF:20\x20\x20\x20\x20\x20\x20\x20\x20Your\x20choice\x20\?\x20\r\n\r\n\r\n\
SF:r\0Change\x20Setup:\r\n\0\x20\x200\x20Server\x20configuration\r\n\r\0\x
SF:20\x206\x20debug\r\n\r\0\x20\x207\x20factory\x20defaults\r\n\r\0\x20\x2
SF:08\x20exit\x20without\x20save\r\n\r\0\x20\x209\x20save\x20and\x20exit\x
SF:20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20Your\x20c
SF:hoice\x20\?\x20\x07\x07\x07\x07\x07\x07\x07\x07\x07\x0720\x07\x07\x07\x
SF:07\x07\x072\x07\x07\x07\x07\x07\x076\x07\x07\x07\x07\r\n\r\n\r\n\r\0Cha
SF:nge\x20Setup:\r\n\0\x20\x200\x20Server\x20configuration\r\n\r\0\x20\x20
SF:6\x20debug\r\n\r\0\x20\x207\x20factory\x20defaults\r\n\r\0\x20\x208\x20
SF:exit\x20without\x20save\r\n\r\0\x20\x209\x20save\x20and\x20exit\x20\x20
SF:\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20Your\x20choice\
SF:x20\?\x20\x07\x07\x07\x072\x07\x07\x07\x07\x07\x07\x07\x07\x07\x071\r\n
SF:\r");
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)==============
SF-Port69-UDP:V=6.40%I=7%D=12/1%Time=547D0F41%P=x86_64-unknown-linux-gnu%r
SF:(DNSStatusRequest,16,"\0\x05\0\x04Invalid\x20Opcode:\x200\0")%r(RPCChec
SF:k,1A,"\0\x05\0\x04Invalid\x20Opcode:\x2029438\0")%r(DNSVersionBindReq,1
SF:6,"\0\x05\0\x04Invalid\x20Opcode:\x206\0")%r(NBTStat,1B,"\0\x05\0\x04In
SF:valid\x20Opcode:\x20-32528\0")%r(Help,1A,"\0\x05\0\x04Invalid\x20Opcode
SF::\x2026725\0")%r(SIPOptions,1A,"\0\x05\0\x04Invalid\x20Opcode:\x2020304
SF:\0");
MAC Address: 00:1D:4B:F0:22:66 (Grid Connect)
No exact OS matches for host (If you know what OS is running on it, see http://nmap.org/submit/ ).
```

```
TCP/IP fingerprint:
OS:SCAN(V=6.40%E=4%D=12/1%OT=80%CT=1%CU=1%PV=Y%DS=1%DC=D%G=Y%M=001D4B%TM=54
OS:7D11DB%P=x86_64-unknown-linux-gnu)SEQ(TS=U)ECN(R=N)T1(R=N)T2(R=N)T3(R=N)
OS:T4(R=N)T5(R=N)T6(R=N)T7(R=N)U1(R=Y%DF=N%T=40%IPL=38%UN=0%RIPL=G%RID=G%RI
OS:PCK=G%RUCK=G%RUD=G)U1(R=N)IE(R=N)

Network Distance: 1 hop

Host script results:
|_dns-brute: Can't guess domain of "172.16.0.106"; use dns-brute.domain script argument.
| firewalk:
| HOP  HOST          PROTOCOL  BLOCKED PORTS
|_0    172.16.0.203  udp       2222,30718,44818
|_ipidseq: Unknown
| qscan:
| PORT   FAMILY  MEAN (us)  STDDEV  LOSS (%)
| 1      0       0.00       -0.00   100.0%
| 69     1       0.00       -0.00   100.0%
| 80     2       0.00       -0.00   100.0%
| 9999   3       0.00       -0.00   100.0%
|_44818  4       0.00       -0.00   100.0%
|_sniffer-detect: Unknown (tests: "________")
| traceroute-geolocation:
|   HOP  RTT   ADDRESS       GEOLOCATION
|_  1    2.12  172.16.0.106  - ,-

TRACEROUTE
HOP RTT     ADDRESS
1   2.12 ms 172.16.0.106

Read data files from: /usr/bin/../share/nmap
OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
# Nmap done at Mon Dec  1 20:11:55 2014 -- 1 IP address (1 host up) scanned in 749.34 seconds
```

Verbose `nmap` fingerprint and enumeration of the L61 PLC

```
# Nmap 6.40 scan initiated Tue Dec  2 22:26:18 2014 as: nmap -e eth0 -oA plc_scan_105 -sC -sS -sU -T4 -A -vv
      -PE -PP -p* --script discovery, enip-info --stats-every 30s 172.16.0.105
Pre-scan script results:
| broadcast-eigrp-discovery:
|_ ERROR: Couldn't get an A.S value.
| broadcast-igmp-discovery:
|   172.16.0.202
|     Interface: eth0
|     Version: 2
|     Group: 224.0.0.251
|     Description: mDNS
|_  Use the newtargets script-arg to add the results as targets
| broadcast-ping:
|   IP: 172.16.0.108  MAC: 00:1d:9c:a3:dc:8d
|   IP: 172.16.0.105  MAC: 00:1d:9c:be:67:93
|_  Use --script-args=newtargets to add the results as targets
| http-icloud-findmyiphone:
|_  ERROR: No username or password was supplied
```

```
| http-icloud-sendmsg:
|_  ERROR: No username or password was supplied
| targets-asn:
|_  targets-asn.asn is a mandatory parameter
| targets-sniffer: Sniffed 6 address(es).
| 172.16.0.105
| 172.16.0.108
| 224.0.0.13
| 224.0.0.1
| 172.16.0.202
|_224.0.0.251
Nmap scan report for 172.16.0.105
Host is up (0.0030s latency).
Scanned at 2014-12-02 22:26:29 EST for 302s
Not shown: 19853 closed ports
PORT      STATE         SERVICE       VERSION
80/tcp    open          http          GoAhead-Webs httpd
| http-backup-finder:
| Spidering limited to: maxdepth=3; maxpagecount=20; withinhost=172.16.0.105
|   http://172.16.0.105/navtree/../..bak
|   http://172.16.0.105/navtree/../../editlimits.asp~
|   http://172.16.0.105/navtree/../. copy./editlimits.asp
|   http://172.16.0.105/navtree/../Copy of ../editlimits.asp
|   http://172.16.0.105/navtree/../Copy (2) of ../editlimits.asp
|   http://172.16.0.105/navtree/../../editlimits.asp.1
|   http://172.16.0.105/navtree/../../editlimits.asp.~1~
|   http://172.16.0.105/navtree/../..bak
|   http://172.16.0.105/navtree/../../diagover.asp~
|   http://172.16.0.105/navtree/../. copy./diagover.asp
|   http://172.16.0.105/navtree/../Copy of ../diagover.asp
|   http://172.16.0.105/navtree/../Copy (2) of ../diagover.asp
|   http://172.16.0.105/navtree/../../diagover.asp.1
|   http://172.16.0.105/navtree/../../diagover.asp.~1~
|   http://172.16.0.105/navtree/../..bak
|   http://172.16.0.105/navtree/../../dataviews.asp~
|   http://172.16.0.105/navtree/../. copy./dataviews.asp
|   http://172.16.0.105/navtree/../Copy of ../dataviews.asp
|   http://172.16.0.105/navtree/../Copy (2) of ../dataviews.asp
|   http://172.16.0.105/navtree/../../dataviews.asp.1
|   http://172.16.0.105/navtree/../../dataviews.asp.~1~
|   http://172.16.0.105/navtree/../..bak
|   http://172.16.0.105/navtree/../../msgconnect.asp~
|   http://172.16.0.105/navtree/../. copy./msgconnect.asp
|   http://172.16.0.105/navtree/../Copy of ../msgconnect.asp
|   http://172.16.0.105/navtree/../Copy (2) of ../msgconnect.asp
|   http://172.16.0.105/navtree/../../msgconnect.asp.1
|   http://172.16.0.105/navtree/../../msgconnect.asp.~1~
|   http://172.16.0.105/navtree/../..bak
|   http://172.16.0.105/navtree/../../serverlog.asp~
|   http://172.16.0.105/navtree/../. copy./serverlog.asp
|   http://172.16.0.105/navtree/../Copy of ../serverlog.asp
|   http://172.16.0.105/navtree/../Copy (2) of ../serverlog.asp
|   http://172.16.0.105/navtree/../../serverlog.asp.1
|   http://172.16.0.105/navtree/../../serverlog.asp.~1~
```

```
|   http://172.16.0.105/navtree/../..bak
|   http://172.16.0.105/navtree/../../backupRestore.html~
|   http://172.16.0.105/navtree/../. copy./backupRestore.html
|   http://172.16.0.105/navtree/../Copy of ../backupRestore.html
|   http://172.16.0.105/navtree/../Copy (2) of ../backupRestore.html
|   http://172.16.0.105/navtree/../../backupRestore.html.1
|   http://172.16.0.105/navtree/../../backupRestore.html.~1~
|   http://172.16.0.105/navtree/../..bak
|   http://172.16.0.105/navtree/../../diagnetwork.asp~
|   http://172.16.0.105/navtree/../. copy./diagnetwork.asp
|   http://172.16.0.105/navtree/../Copy of ../diagnetwork.asp
|   http://172.16.0.105/navtree/../Copy (2) of ../diagnetwork.asp
|   http://172.16.0.105/navtree/../../diagnetwork.asp.1
|_  http://172.16.0.105/navtree/../../diagnetwork.asp.~1~
|_http-chrono: Request times for /index.html; avg: 271.74ms; min: 234.53ms; max: 314.03ms
| http-comments-displayer:
| Spidering limited to: maxdepth=3; maxpagecount=20; withinhost=172.16.0.105
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 141
|     Comment:
|         <!-- * * * CONTROLBUS * * * -->
|
|     Path: http://172.16.0.105/home.asp
|     Line number: 27
|     Comment:
|         <!-- Start tab background -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 237
|     Comment:
|         <!-- * * * BROWSE CHASSIS * * * -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 188
|     Comment:
|         <!--    <a class="entry" onclick="highlightView(this);" onmouseover="overColor(this, '#ffffff');"
      onmouseout="outColor(this, '#bcbcbc');" href="../developersettings.asp" target="home"><img src="../
      images/paper.gif" alt=""> Developer Settings</a>
|         -->
|
|     Path: http://172.16.0.105/css/radevice.css
|     Line number: 90
|     Comment:
|         /* Sortable table headers */
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 191
|     Comment:
|         <!-- * * * DEVICE CONFIGURATION * * * -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 127
|     Comment:
```

```
|         <!-- * * * SOCKET OBJECT * * * -->
|
|     Path: http://172.16.0.105/home.asp
|     Line number: 86
|     Comment:
|         <!-- Begin right side column -->
|
|     Path: http://172.16.0.105/home.asp
|     Line number: 92
|     Comment:
|         <!--
|         <table border=0 width="100" height="100">
|         <tr><td valign="top">
|         <img border="0" src="/images/display/r1c1.bmp"><br>
|         <img src="/images/display/r2c1.bmp"><img name="l0" id="l0" src="/images/display/0.bmp"><img name="
      l1" id="l1" src="/images/display/0.bmp"><img name="l2" id="l2" src="/images/display/0.bmp"><img name="
      l3" id="l3" src="/images/display/0.bmp"><img src="/images/display/r2c6.bmp"><br>
|         <img src="/images/display/r3c1.bmp"><br>
|         <img src="/images/display/r4c1.bmp"><img name="lnet" id="lnet" src="/images/display/led_off.bmp"><
      img name="llink" id="llink" src="/images/display/led_off.bmp"><img name="lmod" id="lmod" src="/images/
      display/led_off.bmp"><img src="/images/display/r4c5.bmp"><br>
|         <img src="/images/display/r5c1.bmp"><br>
|         </td></tr>
|         </table>
|         -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 241
|     Comment:
|         <!-- * * * MESSAGE BOARD * * * -->
|
|     Path: http://172.16.0.105/home.asp
|     Line number: 152
|     Comment:
|         <!-- End body background -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 58
|     Comment:
|         <!-- * * * DATA VIEWS * * * -->
|
|     Path: http://172.16.0.105/home.asp
|     Line number: 9
|     Comment:
|         <!--<script type="text/javascript" language="JavaScript" src="/scripts/XMLDOMWrapper.js"></script>
|           <script type="text/javascript" language="JavaScript" src="/scripts/display.js"></script>
|         -->
|
|     Path: http://172.16.0.105/home.asp
|     Line number: 16
|     Comment:
|         <!-- Start tabs -->
|
|     Path: http://172.16.0.105/home.asp
```

```
|     Line number: 26
|     Comment:
|         <!-- End tabs -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 219
|     Comment:
|         <!-- * * * SERVER MANAGEMENT * * * -->
|
|     Path: http://172.16.0.105/home.asp
|     Line number: 135
|     Comment:
|         <!-- End right side column -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 206
|     Comment:
|         <!-- * * * USER MANAGEMENT * * * -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 245
|     Comment:
|         <!-- * * * EMAIL * * * -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 242
|     Comment:
|         <!--
|           <a class="entry" onclick="highlightView(this);" onmouseover="overColor(this, '#ffffff');"
    onmouseout="outColor(this, '#bcbcbc');" id="/msgboard/index.html" href="../msgboard/index.html" target
    ="home"><img src="../images/paper.gif" alt=""> Message Board</a>
|         -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 176
|     Comment:
|         <!-- * * * ADMINISTRATIVE SETTINGS * * * -->
|
|     Path: http://172.16.0.105/home.asp
|     Line number: 140
|     Comment:
|         <!-- Do not modify below this point -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 109
|     Comment:
|         <!-- * * * NETWORK * * * -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 155
|     Comment:
|         <!-- * * * MISCELLANEOUS * * * -->
|
|     Path: http://172.16.0.105/home.asp
```

```
|     Line number: 33
|     Comment:
|         <!-- Body starts here -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 96
|     Comment:
|         <!-- * * * ETHERNET/IP * * * -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 86
|     Comment:
|         <!-- * * * ADVANCED DIAGNOSTICS * * * -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 71
|     Comment:
|         <!-- * * * DIAGNOSTICS * * * -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 54
|     Comment:
|         <!-- * * * HOME * * * -->
|
|     Path: http://172.16.0.105/navtree/navtree.html
|     Line number: 186
|     Comment:
|_        <!-- * * * DEVELOPER SETTINGS * * * -->
|_http-drupal-modules:
| http-email-harvest:
| Spidering limited to: maxdepth=3; maxpagecount=20; withinhost=172.16.0.105
|_  abtech@prime-controls.com
| http-enum:
|_  /home.asp: Possible admin folder
|_http-favicon: Unknown favicon MD5: D9B704524A6DBEC6E55F47CC295BBB3A
|_http-google-malware: [ERROR] No API key found. Update the variable APIKEY in http-google-malware or set it
      in the argument http-google-malware.api
| http-grep:
|_  ERROR: Argument http-grep.match was not set
| http-headers:
|   Date: FRI JAN 02 02:45:54 1970
|   Server: GoAhead-Webs
|   Last-modified: SUN DEC 30 15:14:14 1979
|   Content-length: 1177
|   Content-type: text/html; charset=utf-8
|   Connection: Close
|
|_  (Request type: GET)
| http-php-version: Logo query returned unknown hash 851c258b740acece42f24de966ada285
|_Credits query returned unknown hash 851c258b740acece42f24de966ada285
| http-sitemap-generator:
|   Directory structure:
|     /
|       asp: 2; html: 2
```

```
|     /css/
|       css: 1
|     /images/
|       gif: 3
|     /navtree/
|       html: 1
|     /scripts/
|       js: 1
|   Longest directory structure:
|     Depth: 1
|     Dir: /css/
|   Total files found (by extension):
|_    asp: 2; css: 1; gif: 3; html: 3; js: 1
| http-title: Rockwell Automation
|_Requested resource was http://172.16.0.105/index.html
| http-vhosts:
|_28 names had status 302
| http-waf-detect: IDS/IPS/WAF detected:
|_172.16.0.105:80/?p4yl04d3=<script>alert(document.cookie)</script>
|_http-wordpress-plugins: nothing found amongst the 100 most popular plugins, use --script-args http-
     wordpress-plugins.search=<number|all> for deeper analysis)
44818/tcp open          EtherNet-IP-2
| enip-info:
|   Vendor: Rockwell Automation/Allen-Bradley (1)
|   Product Name: 1756-EWEB
|   Serial Number: 0x00932471
|   Device Type: Communications Adapter (12)
|   Product Code: 125
|   Revision: 5.1
|_  Device IP: 172.16.0.105
2222/udp  open|filtered msantipiracy
44818/udp open          EtherNet-IP-2
| enip-info:
|   Vendor: Rockwell Automation/Allen-Bradley (1)
|   Product Name: 1756-EWEB
|   Serial Number: 0x00932471
|   Device Type: Communications Adapter (12)
|   Product Code: 125
|   Revision: 5.1
|_  Device IP: 172.16.0.105
MAC Address: 00:1D:9C:BE:67:93 (Rockwell Automation)
Device type: general purpose
Running: Wind River VxWorks
OS CPE: cpe:/o:windriver:vxworks
OS details: VxWorks
TCP/IP fingerprint:
OS:SCAN(V=6.40%E=4%D=12/2%OT=80%CT=1%CU=1%PV=Y%DS=1%DC=D%G=Y%M=001D9C%TM=54
OS:7E8413%P=x86_64-unknown-linux-gnu)SEQ(SP=9A%GCD=1%ISR=A1%TI=I%CI=I%TS=1)
OS:OPS(O1=M5B4NW0NNT11%O2=M5B4NW0NNT11%O3=M5B4NW0NNT11%O4=M5B4NW0NNT11%O5=M
OS:5B4NW0NNT11%O6=M5B4NNT11)WIN(W1=2000%W2=2000%W3=2000%W4=2000%W5=2000%W6=
OS:2000)ECN(R=Y%DF=Y%T=40%W=2000%O=M5B4NW0%CC=N%Q=)T1(R=Y%DF=Y%T=40%S=O%A=S
OS:+%F=AS%RD=0%Q=)T2(R=N)T3(R=Y%DF=Y%T=40%W=2000%S=O%A=S+%F=AS%O=M5B4NW0NNT
OS:11%RD=0%Q=)T4(R=Y%DF=N%T=40%W=2000%S=A%A=Z%F=R%O=%RD=0%Q=)T5(R=Y%DF=N%T=
OS:40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=N%T=40%W=0%S=A%A=Z%F=R%O=%RD=0
```

```
OS:%Q=)T7(R=Y%DF=N%T=40%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N%T=40%IPL=38
OS:%UN=0%RIPL=G%RID=G%RIPCK=Z%RUCK=0%RUD=G)IE(R=N)

Uptime guess: 1.117 days (since Mon Dec  1 19:43:15 2014)
Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=154 (Good luck!)
IP ID Sequence Generation: Incremental

Host script results:
|_dns-brute: Can't guess domain of "172.16.0.105"; use dns-brute.domain script argument.
| firewalk:
| HOP  HOST          PROTOCOL  BLOCKED PORTS
|_0    172.16.0.203  udp       2222
|_ipidseq: Randomized
|_path-mtu: PMTU == 1500
| qscan:
| PORT   FAMILY  MEAN (us)  STDDEV  LOSS (%)
| 1      0       988.80     279.05  0.0%
| 80     1       1393.60    337.47  0.0%
| 44818  1       1514.30    468.07  0.0%
|_44818  1       1381.70    276.78  0.0%
| traceroute-geolocation:
|   HOP  RTT   ADDRESS       GEOLOCATION
|_  1    2.98  172.16.0.105  - ,-

TRACEROUTE
HOP RTT     ADDRESS
1   2.98 ms 172.16.0.105

Read data files from: /usr/bin/../share/nmap
OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
# Nmap done at Tue Dec  2 22:31:31 2014 -- 1 IP address (1 host up) scanned in 313.81 seconds
```

Appendix B. `plcloadtest.py` code

```
#!/usr/bin/python

import os
import struct
import time
import socket
import random
import sys
import select
import logging

import threading
import argparse

IS_AB = True      ## True for Allen-Bradley, False for OMRON
IS_CONTROL = True    ## True for ControlLogix, False for MicroLogix

CIP_SEND_SIZE = 0   ## SEND_SIZE = msg + hdr
CIP_RECV_SIZE = 0   ## RCV_SIZE = msg + hdr
HDR_SEND_LEN = 0
HDR_RCV_LEN = 0

RCV_WAIT = 5

def SetSizes():
  global CIP_SEND_SIZE
  global CIP_RECV_SIZE
  global HDR_SEND_LEN
  global HDR_RCV_LEN

  HDR_SEND_LEN = 66
  HDR_RCV_LEN = 67

  if IS_AB:         ## MICRO
    CIP_SEND_SIZE = 112
    CIP_RECV_SIZE = 144

    if IS_CONTROL:      ## ControlLogix
      CIP_SEND_SIZE = 126
      CIP_RECV_SIZE = 145

  else:         ## OMRON
    CIP_SEND_SIZE = 112
    CIP_RECV_SIZE = 135



class ENIP:

  def connect(self, ip, port, slot):
    self.recvct = 0
    self.error = 0

    log = logging.getLogger('ENIP:Connect')

    status = 0
    self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.sock.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)

    try:
      self.sock.connect((ip, port))
#     self.sock.setblocking(0)
#     self.sock.settimeout(5.0)
      status = 1
    except Exception, e:
      log.error('Status: %i -- Error in connecting to %s:%s \n\tMessage: %s' % (status, ip, port, e))
      return status

    self.register_session()

    self.slot = slot
    self.context = 0

    return status

  def close(self):
    self.sock.close()

  def forwardopen(self):
    self.seqcount = 1
    self.context = 2
    self.conID = self.large_forward_open()

  def send(self, packet):
```

```
    log = logging.getLogger('ENIP:send')
    bytes_sent = 0

    try:
      bytes_sent = self.sock.send(packet)
    except Exception, e:
      log.error('Error sending packet \n\t Message:%s' % e)

    return bytes_sent

  def recv(self):
    ready = select.select([self.sock], [], [], 0)

    if ready[0]:
      response = self.sock.recv(65535)
    else:
      response = None

    # Populate the object with the  data (66+)
#    self.command = response[0:1]
#    self.length = response[2:3]
#    self.shandle = response[4:8]
#    self.status = response[9:12]
#    self.sContext = response[13:21]
#    self.options = response[22:25]
#    self.intHandle = response[26:30]
#    self.timeout = response[31:32]
#    self.iCount = response[33:34]
#    self.typeID1 = response[35:36]
#    self.length1 = response[37:38]
#    self.typeID2 = response[39:40]
#    self.Length2 = response[41:42]
#    self.service = response[43]
#    self.status = response[44:45]
#    self.data = response[46:]

    return response

  def collect(self, obj, tm):
    log = logging.getLogger('ENIP:collect')

    starttime = time.time()
    elapsedtime = 0
    stilltime = True

    exittime = tm + RCV_WAIT    ## Pad the exit time to allow adequate time to collect all responses
    pkt_start = 0
    pkt_end = 0
    tempbuf = ""

##    print "receiving for " + str(exittime) + " seconds..."

    while (stilltime):
      # receving 68 bytes +
      # 66 additional bytes = TCP header (32) + IP header (20) + Ethernet Frame (14)

      data = obj.recv()

##      log.debug('----- LOAD TEST COLLECT : ----- \n\t Recv: %s ' % data)

      if not data is None:
        tempbuf += data

      while not tempbuf is None:
        pkt_start = tempbuf.find('\x6F\x00')

        if pkt_start == -1:
          break

        pkt_end = tempbuf.find('\x6F\x00', pkt_start+2)

        if pkt_end == -1:
          break

        else:
          response = tempbuf[pkt_start:pkt_end-1]
          tempbuf = tempbuf[pkt_end: ]

        log.debug('----- LOAD TEST CHECK: ----- buf %i \t pkt_start %i \t pkt_end %i \n\t Checking response:
              %s' % (len(tempbuf), pkt_start, pkt_end, response.encode('hex')))

        if self.checkpacket(response):
          self.recvct += 1

      time.sleep(0.001)
      elapsedtime = time.time()-starttime
      stilltime = (elapsedtime < exittime)

      if (not stilltime) and (tempbuf):     ## do one more check for a "dangling packet" before exiting
```

```
        pkt_start = tempbuf.find('\x6F\x00')
        if pkt_start != -1:
          log.debug('----- LAST LOAD TEST CHECK: ----- buf %i \t pkt_start %i \t pkt_end %i \n\t Checking
               response: %s' % (len(tempbuf), pkt_start, pkt_end, response.encode('hex')))
          response = tempbuf[pkt_start:len(tempbuf)-1]
          if self.checkpacket(response):
            self.recvct += 1


  def checkpacket(self, response):

    log = logging.getLogger('ENIP:checkpacket')

    if response is None:
      return

    len_response = len(response) + HDR_RCV_LEN
    response = response.encode('hex')


    log.debug('----- LOAD TEST: ----- \n\t Recv: %s %s %s %s %s'
          % (response[0:4], response[4:8], response[8:16], response[16:24], response[24:]))

    if (len_response == CIP_RECV_SIZE):
      if(response[82:86]=='0000'):    # CIP Success message is 0000
##        self.recvct += 1
        log.debug('\t[+] Validated receipt')
        return True
      else:
        log.debug('\t[x] Error message received\n\t %s' % response[82:86])
##        self.error += 1
    else:
      log.debug('\t[x] Unexpected response (length: %i, expected: %i)\n\t %s' % (len_response, CIP_RECV_SIZE
           , response))
##      self.error += 1

    return False


  def wrapUnconnectedSend(self, message, route):
    # Service and meta data
    result = '5202200624010299'.decode('hex')

    # Message req size in bytes (2 bytes)
    result = result + struct.pack('<H', len(message))

    # Message
    result = result + message

    # Length of route (technically only 1 byte, but the other will
    # always be zero anyway
    result = result + struct.pack('<H', len(route)/2)

    result = result + route

    return result

  def getPathOnBack(self, slot):
    #req2 = '0101120f3139322e3136382e3130382e32313000'.decode('hex')
    result = '\x01' + struct.pack('B', slot)

    return result

  def register_session (self):
    log = logging.getLogger('ENIP:reg_session')

    reg = "\x65\x00\x04\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
         x01\x00\x00\x00"
    self.send(reg)

    response = self.sock.recv(1024)

    if response:
      self.session = response[4:8]
    else:
      self.session = None

    response = response.encode('hex')
    reg = reg.encode('hex')

    log.debug('REGISTER SESSION: %i \n\t Mesg: %s %s %s %s %s \n\t Resp: %s %s %s %s %s'
          % (len(response), reg[0:4], reg[4:8], reg[8:16], reg[16:24], reg[24:],
           response[0:4], response[4:8], response[8:16], response[16:24], response[24:]))

    return self.session

  def getAttributeAll(self, cls, inst):
    packet = '\x01\x02\x20' + struct.pack('B', cls) + '\x24' + struct.pack('B', inst)
    return packet
```

```
  def wrapENIPHeader(self, packet, command='\x70\x00'):

    context = struct.pack('<Q', self.context)
    self.context += 1
    leng = struct.pack('<H', len(packet))
    status = '\x00'*4
    options = '\x00\x00\x00\x00'

    header = command + leng + self.session + status + context + options
    return header + packet

  # Assumes CIP
  def wrapENIP(self, itemtype, item, itemtype2=None, item2=None, command='\x70\x00'):

    count = 1

    packet = struct.pack('<H', itemtype)
    packet += struct.pack('<H', len(item))
    packet += item

    if not (itemtype2 is None or item2 is None):
      count = 2
      packet += struct.pack('<H', itemtype2)
      packet += struct.pack('<H', len(item2))
      packet += item2

    # Interface handle and timeout
    packet = '\x00' * 4 + '\x0a\x00' + struct.pack('<H', count) + packet

    return self.wrapENIPHeader(packet, command)

class loadThread (threading.Thread):
  def __init__(self, threadID, tgtIP, tgtPort, duration, rate):
    threading.Thread.__init__(self)
    self.name = threadID
    self.tgtIP = tgtIP
    self.tgtPort = tgtPort
    self.duration = duration
    self.rate = rate
    self.totals = (0, 0, 0)

  def run(self):

#   log = logging.FileHandler(self.name+'_verbose.log')
#   log.setLevel(logging.DEBUG)
#   logging.getLogger('').addHandler(log)
    log = logging.getLogger('loadThread:run')

    log.debug('Starting thread %s' % self.name)
    # Get lock to synchronize threads
#   threadLock.acquire()
    self.tstart = time.time()
    self.totals = LoadTest(self.tgtIP, self.tgtPort, self.duration, self.rate)
    self.tfinish = time.time()
    ttime = self.tfinish-self.tstart
    self.totals += (ttime - RCV_WAIT, )   ## Ignore the 5 second RECV() wait time

    log.info('   [+] Thread %s done in %.4f sec \n\t #err: %i \t send: %.2f \t recv: %.2f (bytes)'
         % (self.name, self.totals[3], self.totals[0], self.totals[1], self.totals[2]))

    # Free lock to release next thread
#   threadLock.release()

  def join(self):
    threading.Thread.join(self)
    return self.totals

def sendPacket(ip, port, cls, att):
  e = ENIP()

  e.connect(ip,port,0)

  # CIP portion
  d = e.getAttributeAll(cls, att)

  # ENIP header
  d = e.wrapENIP(0, '', 0xB2, d, command='\x6f\x00')

  e.send(d)
  reply = e.recv().encode('hex')
# reply = e.recv()

  e.close()
  return reply

def LoadTest(tgtIP, tgtPort, tm, pps):

  log = logging.getLogger('main-LoadTest')
```

```
  s_total = 0
  r_total = 0
  error = 0
  send = 0
  recvct = 0
  sendttl = 0

  obj = ENIP()
  if not obj.connect(tgtIP, tgtPort, 0):
    log.debug('No connection. Breaking')
    return (0,0,0)

  # CIP portion
  frame = obj.getAttributeAll(1,1)

  # ENIP header
  if IS_CONTROL:
    frame = obj.wrapUnconnectedSend(frame, obj.getPathOnBack(0))

  frame = obj.wrapENIP(0, '', 0xB2, frame, command='\x6f\x00')

  frame_hex = frame.encode('hex')
  if pps == 0:    # Unlimited packet rate
    pps = 1000000

  pkt_sleep = 1.0 / pps

  rcv = threading.Thread(target=obj.collect, args=(obj, tm))
  rcv.start()
  # time.sleep(5)

  starttime = time.time()
  curr = time.time()

  while curr < (starttime + tm):
    sec = curr + 1
    sendct = 0
    sleeptime = pkt_sleep

    while (curr <= sec):

         if (sendct < pps):
        # Sending 46 bytes / receive 70 in return, plus
        # 66 additional bytes = TCP header (32) + IP header (20) + Ethernet Frame (14)

          pkt_start = time.time()

          while ((time.time() - pkt_start) < sleeptime):
            #log.debug('Sleeping it off..... START %.5f / NOW %.5f / UNTIL %.5f' % (pkt_start, ((time.time()
                  - pkt_start)), sleeptime))
            pass

          send = obj.send(frame) + HDR_SEND_LEN

          log.debug('----- LOAD TEST: ----- TIME\t%.5f \n\t Sent: %s %s %s %s %s'
              % (pkt_start, frame_hex[0:4], frame_hex[4:8], frame_hex[8:16], frame_hex[16:24], frame_hex
                   [24:]))


          if send != CIP_SEND_SIZE:
            log.debug('\t[X] Error sending\n\t (sent: %i, expected: %i)' % (send, CIP_SEND_SIZE))
            error += 1
            sendct += 1
          else:
            s_total += send
            sendct += 1
            sendttl += 1

            pkt_fin = time.time() - pkt_start

          sleeptime = pkt_sleep - pkt_fin

          #print 'time.time', time.time(), 'PKT_ST', pkt_start, 'PKT_FIN:', pkt_fin

          if(sleeptime >= 0.0):
            time.sleep(sleeptime*.1)

         curr = time.time()

##  log.info('waiting for recv() threads to close...')
  rcv.join()

  recvct += obj.recvct

##  print "Leaving LoadTest ( Errors: ", error, "Sent Ct: ", sendttl, " / Recv Ct: ", recvct, ")"
  error += obj.error + (sendttl - obj.recvct)       ## Errors are Invalid responses + No responses

  r_total = recvct * CIP_RECV_SIZE    ## Successful receives count * response size (need to verify)
```

```
  obj.close()

##  print "Leaving LoadTest (Err: ", error, " / Sent: ", s_total, " / Recv: ", r_total, ")"
  return(error, s_total, r_total)

def FindMaxSessions(tgtIP, tgtPort):
  log = logging.getLogger('main-FindMaxSessions')

  conn = []
  i = 0

  while True:
    i += 1
    obj = ENIP()
    obj.connect(tgtIP, tgtPort, 0)
    if (not obj.session):
      log.debug('\t[X] No Session Received....')
      break
    conn.append(obj)
    log.debug('Registering session #: %i \tID: %s' % (i, obj.session.encode('hex')))

#   time.sleep(5)
#
#  for i in range(0, 10):
#    obj = conn.pop()
#    print "[Closing 10 Connections]\tSession #: ", i+1, "\tID: ", obj.session.encode('hex')
#    obj.close()
#
#  time.sleep(5)
#  i = 0
#
#  while True:
#    i += 1
#    obj = ENIP()
#    obj.connect(tgtIP, tgtPort, 0)
#    if not obj.session:
#      print "No session received..."
#      break
#    conn.append(obj)
#    print "[Building list 2]\tSession #: ", i, "\tID: ", obj.session.encode('hex')
#


  maxlength = len(conn)

  time.sleep(5)

  for i in range(0, len(conn)):
    obj = conn.pop()
    log.debug('\t Closing session #: %i \tID: %s' % (i+1, obj.session.encode('hex')))
    obj.close()

  return maxlength

threadLock = threading.Lock()
threads = []

def getArgs():
  parser = argparse.ArgumentParser(
                  description="A load testing program for Allen Bradely L61 Controller",
                 )
  ## Required arguments
  parser.add_argument("tgtIP", help="IP Address of target (proxied) device")
  parser.add_argument("tgtPort", help="TCP Port of target (proxied) device", type=int)

  ## Optional arguments
  parser.add_argument("-f","--filename", help="Base file name for logging (will append _verbose.log and
       _console.log, default: benchtest", type=str)
  parser.add_argument("-d", "--duration", help="Duration of each test (seconds), default: 60s", type=int)
  parser.add_argument("-r","--rate", help="Rate of packets per second to send , default: 50pps", type=int)
  parser.add_argument("-t","--min-threads", help="Number of threads to start testing, default: 1", type=int)
  parser.add_argument("-T","--max-threads", help="Number of threads to test up to, default: MAX", type=int)
  parser.add_argument("-v","--variation", help="Type of equipment under test ABC=ControlLogix ABM=MicroLogix
        or OM=OMRON", type=str)


  args = parser.parse_args()

  if args.duration:
    duration = args.duration
  else:
    duration = 60

  if args.rate:
    rate = args.rate
  else:
    rate = 50
```

```
  if args.min_threads:
    threads_min = args.min_threads
  else:
    threads_min = 1

  if args.max_threads:
    threads_max = args.max_threads
  else:
    threads_max = 0

  if args.filename:
    filename = args.filename
  else:
    filename = "benchtest"

  global IS_AB
  global IS_CONTROL

  if args.variation:
    if args.variation == 'ABC':
      IS_AB = True
      IS_CONTROL = True
    elif args.variation == 'ABL':
      IS_AB = True
      IS_CONTROL = False
    elif args.variation == 'OM':
      IS_AB = False
      IS_CONTROL = False

  return args.tgtIP, args.tgtPort, args.duration, args.rate, threads_min, threads_max, filename


def main():

  ##  tgtIP = '192.168.0.119'
  ##  tgtPort = 44818
  ##  cls = 1
  ##  att = 1

  tgtIP, tgtPort, duration, rate, threads_min, threads_max, logfile = getArgs()

  cls = 1  # CIP Class
  att = 1  # CIP Attribute

  runtime = time.time()

  logging.basicConfig(level=logging.INFO,
            format='%(asctime)s %(name)-25s %(levelname)-8s %(message)s',
##            datefmt='%d-%b-%y %H:%M:%S',
            filename=logfile+'_verbose.log',
            filemode='w')

  console = logging.StreamHandler()
  console.setLevel(logging.INFO)

  file = logging.FileHandler(logfile+'_console.log')
  file.setLevel(logging.INFO)

  formatter = logging.Formatter('%(message)s')
  console.setFormatter(formatter)
  file.setFormatter(formatter)
  logging.getLogger('').addHandler(console)
  logging.getLogger('').addHandler(file)

  log = logging.getLogger('main-main')

  SetSizes()
  ## Start tcpdump HERE
  log.info('Testing started at %s' % time.asctime(time.localtime(time.time())))

  log.info('\tIS_AB: %s / IS_CONTROL: %s \t CIP_SEND_SIZE: %i / CIP_RECV_SIZE %i / HDR_SEND_LEN %i /
       HDR_RCV_LEN %i' % (IS_AB, IS_CONTROL, CIP_SEND_SIZE, CIP_RECV_SIZE, HDR_SEND_LEN, HDR_RCV_LEN))
#### Prerequisite: Verify connectivity ################################################
##  log.info('>> Verifying connectivity with sendPacket...')
##
##    #    self.command = response[0:4]   ENIP Header
##    #    self.length = response[4:8]
##    #    self.shandle = response[8:16]
##    #    self.status = response[16:24
##    #    self.sContext = response[24:40]
##    #    self.options = response[40:48]
##    #    self.intHandle = response[48:56]
##    #    self.timeout = response[56:60]
##    #    self.iCount = response[60:64]
##    #    self.typeID1 = response[64:68]
##    #    self.length1 = response[68:72]
##    #    self.typeID2 = response[72:76]
##    #    self.Length2 = response[76:80]
##    #    self.service = response[80:82] CIP Message
```

```
##    #    self.status = response[82:86]
##    #    self.data = response[86:]
##
##  packet = sendPacket(tgtIP, tgtPort, cls, att)
##  log.debug('Good message: \n\t [ENIP]\t %s %s %s %s %s \n\t [CIP]\t %s %s %s' % \
##        (packet[0:4], packet[4:8], packet[8:16], packet[16:24], packet[24:80], \
##        packet[80:82], packet[82:86], packet[86:]))
##
##  packet = sendPacket(tgtIP, tgtPort, cls, 16)
##  log.debug('Error message: \n\t [ENIP]\t %s %s %s %s %s \n\t [CIP]\t %s %s %s' % \
##        (packet[0:4], packet[4:8], packet[8:16], packet[16:24], packet[24:80], \
##        packet[80:82], packet[82:86], packet[86:]))
##
###### Test 1: Find max sessions ############################################################
##  log.info('>> Running FindMaxSessions...')
##  maxsessions = FindMaxSessions(tgtIP, tgtPort)
##
##  log.info('\t Maximum sessions: \t %i' % maxsessions)
##
#### Test 2: Find max throughput in a session ##########################################
# duration = 60   # length of test in seconds
# rate = 50    # packets per second (0 = MAX; 62-65 = Effective MAX)
# totals = (0, 0, 0)
#
# log.info('>> Running single instance of LoadTest for %i seconds (@ %i pps)...' % (duration, rate))
#
# totals = LoadTest(tgtIP, tgtPort, duration, rate)
# log.info('\t Errors: \t\t %i \n\t Avg rate (send): \t %.2f bytes/sec \n\t Avg rate (recv): \t %.2f bytes/
    sec' \
#        % (totals[0], (totals[1]/duration), (totals[2]/duration)))
#
## Test 3: Find max throughput with multiple sessions ###############################

#   numthreads = 3  # number of concurrent threads

  if threads_max == 0: threads_max = maxsessions  # number of concurrent threads
  totalstable =[]
  extime = 0

  for j in range(threads_min, threads_max+1):
    totals = (0, 0, 0, 0, 0)

    log.info('>> Starting threaded LoadTest with %i / %i threads for %i seconds (@ %i pps)...' \
        % (j, threads_max, duration, rate))
    starttime = time.time()

    # Create new threads
    for i in range (1, j+1):
      thrID = logfile + '_' + `j` + '.' + `i`
      thread = loadThread(thrID, tgtIP, tgtPort, duration, rate)
      thread.start()
      threads.append(thread)
      wait = 0.1 * random.random()
      log.info('[*] Stagger time for thread %i: %.3f sec' % (i, wait))
      time.sleep(wait)

    # Wait for all threads to complete

    for t in threads:
      returned = t.join()
      totals = (totals[0] + returned[0], totals[1] + returned[1], totals[2] + returned[2], totals[3] +
          returned[3])
    extime = time.time()-starttime

    totals += (extime, )

    totalstable.append(totals)

    log.info('   --- Round %i summary ------------------ \n\t Elapsed Time: \t\t %.4f sec \n\t Errors: \t\t
        %i \n\t Avg rate (send): \t %.2f bytes/sec \n\t Avg rate (recv): \t %.2f bytes/sec \n\t Avg thread
        time: \t %.4f sec'
        % (j, extime,
         totals[0],
         (totals[1]/(totals[3]/j)),
         (totals[2]/(totals[3]/j)),
         totals[3]/j))

#   log.debug('Thread pool BEFORE: \t %s' % threads)

    # Empty the list of threads to avoid residual data
    while len(threads) > 0 : threads.pop()

 #     log.debug('Thread pool AFTER: \t %s' % threads)

  log.info('Printing tabulated results to screen.....')

  print "\nThr\tDur\tRt\tErr\tPktSnd\tPktRcv\tAvgSendRt\tAvgRecvRt\tAvgThTime\tElTime"
  print "---\t---\t--\t---\t------\t------\t---------\t---------\t---------\t------"
```

```
  for j in range(0, 1+(threads_max-threads_min)):
    print j+threads_min, "\t", duration, "\t", rate, "\t", totalstable[j][0], "\t", (totalstable[j][1]/
          CIP_SEND_SIZE), "\t", (totalstable[j][2]/CIP_RECV_SIZE),"\t",(totalstable[j][1]/(totalstable[j
          ][3]/(j+threads_min))), "\t", (totalstable[j][2]/(totalstable[j][3]/(j+threads_min))), "\t", (
          totalstable[j][3]/(j+threads_min)), "\t", totalstable[j][4]

  endtime = time.time()

  log.info('Load test completed at %s -- Total elapsed time: %i sec' % (time.asctime(time.localtime(time.
       time())), endtime-runtime))

  ## Stop tcpdump HERE

if __name__ == '__main__':
  main()
```

Appendix C. PLC Templates

L61.conf: Allen-Bradley L61 PLC

```
#########################################################################
#... DEFAULT TEMPLATE ..................................................
# honeyd will dynamically clone the default template for any IP
# address on its subnet that is not specifically bound to a template
# (below)
#########################################################################
create default
set default personality "VxWorks"
set default ethernet "Rockwell Automation"
# set default default icmp action open
set default default tcp action reset
add default tcp port 80 proxy 172.16.0.105:80
add default tcp port 44818 proxy 172.16.0.105:44818
add default udp port 44818 proxy 172.16.0.105:44818
add default udp port 2222 proxy 172.16.0.105:2222
set default default udp action block

#########################################################################
#... INVIS TEMPLATE ....................................................
# the invis template is used to exclude other hosts (e.g. production hosts)
# that already exist on the network.
#########################################################################
create invis
set invis default tcp action block
set invis default udp action block
set invis default icmp action block
bind 172.16.0.105 invis
bind 172.16.0.72 invis



#########################################################################
#... ADDITIONAL TEMPLATES ..............................................
# Additional unique templates can be defined here. Multiple IP
# addresses can be bound to a single template
#########################################################################
create L61
set L61 personality "VxWorks"
set L61 ethernet "Rockwell Automation"
set L61 default tcp action reset
set L61 default udp action reset
set L61 default icmp action open
add L61 tcp port 80 proxy 172.16.0.105:80
add L61 tcp port 44818 proxy 172.16.0.105:44818
add L61 udp port 44818 proxy 172.16.0.105:44818
add L61 udp port 2222 proxy 172.16.0.105:2222
bind 172.16.108.2 L61
bind 172.16.108.3 L61
bind 172.16.108.4 L61
bind 172.16.108.5 L61
....
bind 172.16.108.74 L61
bind 172.16.108.75 L61
bind 172.16.108.76 L61


#########################################################################
#... ICSPROXY options ...................................................
# The options for the ICSPROXY functionality are split into a separate
# file, for clarity.
#########################################################################
## honeyd configuration parsing uses these REGEX to cast types to options (STRING|FLOAT|INT [NUMBER])
##     [0-9]+    { yylval.number = atoi(yytext); return NUMBER; }
##     [0-9]+\.[0-9]+  { yylval.floatp = atof(yytext); return FLOAT; }
##     [\$A-Za-z][\.\(\)\/A-Za-z_\-0-9\*]* { yylval.string = strdup(yytext); return STRING; }
#########################################################################

#########################################################################
## ... COMPUTED OPTIONS .................................................
## The following options are "find" criteria, and the associated
## "replace" terms are computed using template-specific properties when
## the search list is populated into each template.
#########################################################################
## 'ip' is a keyword, so use ipaddr instead.
## Force the interpretation of the dotted decimal notation as a string with a "$" at the beginning
option icsproxy ipaddr $172.16.0.105

## 'ethernet' is a keyword, so use ethaddr instead.
## Force the interpretation of the dotted decimal notation as a string with a "$" at the beginning.
```

```
## Also use dashes (-) instead of colons (:) to separate the MAC address
option icsproxy ethaddr $00-1D-9C-BE-67-93

option icsproxy hostname $eWeb_xxx

option icsproxy serial $00932471

# Pass binary patterns with the prefix $(hex)
option icsproxy $(hex)313735362d45574542 $(hex)3534313900393324716768
# option icsproxy      1 7 5 6 - E W E B        5 4 1 9\x 9 3 $ q C D


######################################################################
## ... ADDITIONAL OPTIONS ............................................
## Additional [static] find/replace criteria can be specified using the
## following syntax:
##     OPTION ICSPROXY <FIND> <REPLACE>
######################################################################
# option icsproxy plc.water.s5 plc.water.A0

# option icsproxy site facility
# option icsproxy interest bail


option dummy test1 test2
option dummy test3 test4
option dummy test5 test6
option check test1 test2
option check test3 test4
option check test5 test6
```

CP1L.conf: Omron CP1L PLC

```
######################################################################
#... DEFAULT TEMPLATE ...............................................
# honeyd will dynamically clone the default template for any IP
# address on its subnet that is not specifically bound to a template
# (below)
######################################################################
create default
set default personality "VxWorks"
set default ethernet "Rockwell Automation"
# set default default icmp action open
set default default tcp action reset
add default tcp port 80 proxy 172.16.0.105:80
add default tcp port 44818 proxy 172.16.0.105:44818
add default udp port 44818 proxy 172.16.0.105:44818
add default udp port 2222 proxy 172.16.0.105:2222
set default default udp action block

######################################################################
#... INVIS TEMPLATE .................................................
# the invis template is used to exclude other hosts (e.g. production hosts)
# that already exist on the network.
######################################################################
create invis
set invis default tcp action block
set invis default udp action block
set invis default icmp action block
bind 172.16.0.105 invis
bind 172.16.0.72 invis


######################################################################
#... ADDITIONAL TEMPLATES ...........................................
# Additional unique templates can be defined here. Multiple IP
# addresses can be bound to a single template
######################################################################
create CP1L
set CP1L personality "HP iLO 2 remote management interface"
set CP1L ethernet "Grid Connect"
set CP1L default tcp action reset
set CP1L default udp action reset
set CP1L default icmp action open
add CP1L tcp port 80 proxy 172.16.0.106:80
add CP1L tcp port 9999 proxy 172.16.0.106:9999
add CP1L tcp port 44818 proxy 172.16.0.106:44818
add CP1L udp port 44818 proxy 172.16.0.106:44818
add CP1L udp port 69 proxy 172.16.0.106:69
add CP1L udp port 2222 proxy 172.16.0.106:2222
add CP1L udp port 9600 proxy 172.16.0.106:9600
add CP1L udp port 30718 proxy 172.16.0.106:30718
add CP1L udp port 44818 proxy 172.16.0.106:44818
bind 172.16.108.2 CP1L
```

```
bind 172.16.108.3 CP1L
bind 172.16.108.4 CP1L
...
bind 172.16.108.74 CP1L
bind 172.16.108.75 CP1L
bind 172.16.108.76 CP1L

##########################################################################
#... ICSPROXY options ..................................................
# The options for the ICSPROXY functionality are split into a separate
# file, for clarity.
##########################################################################
## honeyd configuration parsing uses these REGEX to cast types to options (STRING|FLOAT|INT [NUMBER])
##     [0-9]+    { yylval.number = atoi(yytext); return NUMBER; }
##     [0-9]+\.[0-9]+  { yylval.floatp = atof(yytext); return FLOAT; }
##     [\$A-Za-z][\.\(\)\/A-Za-z_\-0-9\*]* { yylval.string = strdup(yytext); return STRING; }
##########################################################################

##########################################################################
## ... COMPUTED OPTIONS ..................................................
## The following options are "find" criteria, and the associated
## "replace" terms are computed using template-specific properties when
## the search list is populated into each template.
##########################################################################
## 'ip' is a keyword, so use ipaddr instead.
## Force the interpretation of the dotted decimal notation as a string with a "$" at the beginning
option icsproxy ipaddr $172.16.0.106

## 'ethernet' is a keyword, so use ethaddr instead.
## Force the interpretation of the dotted decimal notation as a string with a "$" at the beginning.
## Also use dashes (-) instead of colons (:) to separate the MAC address
option icsproxy ethaddr $00-1D-4B-F0-22-66

option icsproxy hostname $CP1W-EIP61

option icsproxy serial $4bf02266

# Pass binary patterns with the prefix $(hex)
# option icsproxy     C P 1 W - E I P 6 1        R J 5 X ! N Q A 9 2
option icsproxy $(hex)435031572d4549503631 $(hex)524a3558214e51413932

##########################################################################
## ... ADDITIONAL OPTIONS ................................................
## Additional [static] find/replace criteria can be specified using the
## following syntax:
##     OPTION ICSPROXY <FIND> <REPLACE>
##########################################################################
# option icsproxy plc.water.s5 plc.water.A0

# option icsproxy site facility
# option icsproxy interest bail


option dummy test1 test2
option dummy test3 test4
option dummy test5 test6
option check test1 test2
option check test3 test4
option check test5 test6
```

Appendix D. Performance Test Raw Data

Table 4.1. Summary Statistics (verbose) for the L61, EtherNet/IP Protocol

Platform (ENIP)	rate	Repetition	Mean	StdDev	Var	Min	Max
Baseline-L61	10x30	run1	29.46%	1.71%	0.03%	26.19%	33.74%
		run2	29.80%	1.76%	0.03%	24.97%	36.30%
		run3	32.82%	1.39%	0.02%	29.58%	35.64%
	12x25	run1	35.09%	1.44%	0.02%	31.89%	37.88%
		run2	35.65%	2.34%	0.05%	32.33%	49.03%
		run3	38.57%	1.61%	0.03%	35.51%	45.00%
	20x15	run1	58.69%	1.63%	0.03%	55.36%	62.82%
		run2	59.13%	1.88%	0.04%	55.28%	63.77%
		run3	62.20%	1.96%	0.04%	55.76%	65.77%
	4x75	run1	9.28%	1.42%	0.02%	4.48%	15.37%
		run2	9.09%	0.83%	0.01%	7.09%	11.06%
		run3	9.22%	1.13%	0.01%	6.22%	12.58%
	5x60	run1	14.41%	1.86%	0.03%	11.18%	22.78%
		run2	14.19%	1.48%	0.02%	11.41%	20.86%
		run3	14.71%	1.14%	0.01%	13.00%	17.44%
PC-L61	10x30	run1	30.06%	1.88%	0.04%	26.39%	34.82%
		run2	30.93%	1.79%	0.03%	27.42%	34.64%
		run3	34.44%	1.86%	0.03%	30.96%	39.34%
	12x25	run1	36.18%	1.68%	0.03%	32.80%	40.84%
		run2	36.72%	1.64%	0.03%	33.71%	40.38%
		run3	40.58%	1.72%	0.03%	36.13%	43.03%
	20x15	run1	63.86%	1.82%	0.03%	58.39%	68.33%
		run2	63.95%	1.66%	0.03%	60.09%	67.68%
		run3	67.80%	1.95%	0.04%	62.72%	72.40%
	4x75	run1	11.37%	2.15%	0.05%	9.52%	23.74%
		run2	11.48%	2.28%	0.05%	9.44%	23.40%
		run3	11.40%	1.35%	0.02%	8.72%	18.20%
	5x60	run1	16.36%	1.53%	0.02%	14.00%	23.34%
		run2	16.14%	0.99%	0.01%	14.30%	18.99%
		run3	16.51%	1.11%	0.01%	14.48%	18.80%
Pi-L61	10x30	run1	27.45%	1.98%	0.04%	23.73%	33.84%
		run2	28.06%	1.85%	0.03%	23.00%	31.52%
		run3	31.82%	3.08%	0.09%	28.92%	50.50%
	12x25	run1	33.44%	1.35%	0.02%	29.20%	36.06%
		run2	33.71%	1.62%	0.03%	29.10%	37.81%
		run3	36.98%	1.61%	0.03%	33.77%	40.13%
	20x15	run1	60.06%	2.45%	0.06%	55.78%	71.80%
		run2	59.85%	1.59%	0.03%	55.06%	62.80%
		run3	63.34%	2.30%	0.05%	54.71%	67.00%
	4x75	run1	7.88%	1.07%	0.01%	5.32%	10.28%
		run2	7.65%	1.19%	0.01%	4.18%	10.62%
		run3	7.58%	0.97%	0.01%	5.16%	10.00%
	5x60	run1	12.65%	1.37%	0.02%	10.69%	15.48%
		run2	12.44%	1.07%	0.01%	10.11%	15.58%

Table 4.2. Summary Statistics (verbose) for the L61, HTTP Protocol

Platform (ENIP)	rate	Repetition	Mean	StdDev	Var	Min	Max
Baseline-L61	10x5	run1	18.03%	1.31%	0.02%	15.87%	22.33%
		run2	17.50%	1.29%	0.02%	14.47%	20.00%
		run3	22.63%	3.07%	0.09%	17.27%	30.60%
	1x50	run1	0.66%	4.64%	0.22%	0.00%	32.80%
		run2	0.64%	4.53%	0.20%	0.00%	32.00%
		run3	0.69%	4.86%	0.24%	0.00%	34.40%
	25x2	run1	54.09%	2.22%	0.05%	46.33%	57.20%
		run2	54.19%	2.31%	0.05%	46.60%	58.07%
		run3	59.16%	1.95%	0.04%	50.80%	62.93%
	2x25	run1	0.09%	0.09%	0.00%	0.00%	0.47%
		run2	0.14%	0.21%	0.00%	0.00%	1.20%
		run3	0.21%	0.64%	0.00%	0.00%	4.47%
	5x10	run1	14.12%	1.48%	0.02%	10.67%	17.47%
		run2	13.48%	1.91%	0.04%	7.27%	17.53%
		run3	12.52%	1.50%	0.02%	9.00%	15.07%
PC-L61	10x5	run1	32.28%	0.11%	0.00%	32.07%	32.53%
		run2	32.27%	0.13%	0.00%	31.80%	32.47%
		run3	32.21%	0.16%	0.00%	31.73%	32.53%
	1x50	run1	32.41%	0.09%	0.00%	32.13%	32.53%
		run2	32.43%	0.11%	0.00%	32.00%	32.53%
		run3	32.44%	0.06%	0.00%	32.33%	32.53%
	25x2	run1	58.53%	1.32%	0.02%	54.80%	60.67%
		run2	58.44%	1.50%	0.02%	51.87%	60.87%
		run3	63.11%	1.51%	0.02%	57.67%	66.27%
	2x25	run1	32.47%	0.07%	0.00%	32.27%	32.53%
		run2	32.47%	0.08%	0.00%	32.13%	32.53%
		run3	32.33%	0.11%	0.00%	32.07%	32.53%
	5x10	run1	32.34%	0.11%	0.00%	32.13%	32.53%
		run2	32.33%	0.12%	0.00%	32.07%	32.53%
		run3	32.30%	0.11%	0.00%	32.07%	32.53%
Pi-L61	10x5	run1	31.87%	0.24%	0.00%	31.33%	32.27%
		run2	31.91%	0.22%	0.00%	31.33%	32.33%
		run3	31.74%	0.19%	0.00%	31.40%	32.20%
	1x50	run1	32.11%	0.18%	0.00%	31.67%	32.47%
		run2	32.05%	0.21%	0.00%	31.47%	32.47%
		run3	32.10%	0.16%	0.00%	31.73%	32.40%
	25x2	run1	57.35%	1.93%	0.04%	51.27%	64.00%
		run2	57.42%	1.31%	0.02%	53.80%	60.33%
		run3	62.13%	1.57%	0.02%	56.20%	65.07%
	2x25	run1	32.24%	0.14%	0.00%	31.87%	32.47%
		run2	32.25%	0.16%	0.00%	31.73%	32.47%
		run3	32.24%	0.14%	0.00%	31.80%	32.53%
	5x10	run1	31.95%	0.16%	0.00%	31.67%	32.47%
		run2	31.95%	0.23%	0.00%	31.40%	32.27%

Table 4.3. Summary Statistics (verbose) for the CP1L, ENIP Protocol

Platform (ENIP)	rate	Repetition	Mean	StdDev	Var	Min	Max
Baseline-CP1L	1x80	run1	0.00%	0.00%	0.00%	0.00%	0.00%
		run2	0.00%	0.00%	0.00%	0.00%	0.00%
		run3	0.00%	0.00%	0.00%	0.00%	0.00%
	2x40	run1	3.68%	13.98%	1.95%	0.00%	75.04%
		run2	2.43%	8.80%	0.77%	0.00%	45.83%
		run3	4.23%	18.01%	3.25%	0.00%	92.00%
	4x20	run1	13.83%	30.08%	9.05%	0.29%	99.21%
		run2	11.52%	22.99%	5.28%	0.33%	99.17%
		run3	7.78%	16.02%	2.57%	0.29%	75.79%
PC-CP1L	1x80	run1	0.00%	0.00%	0.00%	0.00%	0.00%
		run2	0.00%	0.00%	0.00%	0.00%	0.00%
		run3	0.00%	0.00%	0.00%	0.00%	0.00%
	2x40	run1	0.17%	0.11%	0.00%	0.00%	0.46%
		run2	0.17%	0.13%	0.00%	0.00%	0.54%
		run3	0.19%	0.15%	0.00%	0.00%	0.63%
	4x20	run1	28.35%	4.40%	0.19%	23.38%	52.92%
		run2	28.34%	4.25%	0.18%	23.58%	50.96%
		run3	28.39%	4.90%	0.24%	24.71%	51.04%
Pi-CP1L	1x80	run1	0.00%	0.00%	0.00%	0.00%	0.00%
		run2	0.00%	0.00%	0.00%	0.00%	0.00%
		run3	0.00%	0.00%	0.00%	0.00%	0.00%
	2x40	run1	3.09%	10.43%	1.09%	0.00%	49.71%
		run2	1.32%	5.94%	0.35%	0.00%	42.33%
		run3	2.80%	9.96%	0.99%	0.00%	58.63%
	4x20	run1	29.36%	9.20%	0.85%	22.92%	87.04%
		run2	29.87%	7.77%	0.60%	24.33%	67.33%
		run3	29.31%	6.49%	0.42%	23.83%	60.88%

Table 4.4. Summary Statistics (verbose) for the CP1L, HTTP Protocol

Platform (ENIP)	rate	Repetition	Mean	StdDev	Var	Min	Max
Baseline-CP1L	1x20	run1	87.59%	0.45%	0.00%	86.67%	88.50%
		run2	87.68%	0.42%	0.00%	86.50%	88.50%
		run3	87.71%	0.44%	0.00%	86.83%	88.50%
	2x10	run1	85.83%	0.83%	0.01%	83.50%	87.33%
		run2	85.74%	0.77%	0.01%	83.00%	87.67%
		run3	85.75%	0.70%	0.00%	84.33%	87.50%
	4x5	run1	86.76%	0.63%	0.00%	85.33%	88.17%
		run2	86.79%	0.60%	0.00%	85.83%	88.33%
		run3	86.78%	0.64%	0.00%	85.33%	87.83%
PC-CP1L	1x20	run1	88.66%	0.02%	0.00%	88.50%	88.67%
		run2	88.65%	0.05%	0.00%	88.50%	88.67%
		run3	88.66%	0.05%	0.00%	88.50%	88.83%
	2x10	run1	88.88%	0.18%	0.00%	88.67%	89.33%
		run2	88.87%	0.16%	0.00%	88.67%	89.33%
		run3	88.86%	0.18%	0.00%	88.67%	89.17%
	4x5	run1	88.69%	0.06%	0.00%	88.67%	88.83%
		run2	88.66%	0.05%	0.00%	88.50%	88.83%
		run3	88.69%	0.08%	0.00%	88.50%	89.00%
Pi-CP1L	1x20	run1	88.73%	0.32%	0.00%	88.50%	90.83%
		run2	88.79%	0.52%	0.00%	88.50%	91.50%
		run3	88.66%	0.08%	0.00%	88.50%	88.83%
	2x10	run1	89.15%	0.22%	0.00%	88.83%	90.17%
		run2	89.08%	0.18%	0.00%	88.83%	89.50%
		run3	89.18%	0.30%	0.00%	88.67%	90.67%
	4x5	run1	89.42%	2.66%	0.07%	88.50%	100.00%
		run2	88.69%	0.13%	0.00%	88.50%	89.33%
		run3	88.68%	0.18%	0.00%	88.50%	89.83%

Bibliography

1. M. Bailey et al. "The Blaster Worm: Then and Now," *IEEE Security & Privacy* , vol.3, no.4, pp. 26-31, July-August 2005.

2. D. Berman, "Emulating Industrial Controls System Devices Using Gumstix Technology," M.S. Thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, 2012. (www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA563104)

3. R. Bodenheim et al., "Evaluation of the Ability of the Shodan Search Engine to Identify Internet-Facing Industrial Control Devices," *International Journal of Critical Infrastructure Protection*, 7.2, pp. 114-123, 2014.

4. S. Boyer, *SCADA: Supervisory Control and Data Acquisition*, 4th ed. International Society of Automation, 2009.

5. CERT. (January 2002). "CERT Advisory CA-2001-19 'Code Red' Worm Exploiting Buffer Overflow in IIS Indexing Service DLL," Cert.org. [Online] Available: http://www.cert.org/historical/advisories/ca-2001-19.cfm. [Accessed January 2015].

6. CERT. (January 2003). "CERT Advisory CA-2003-04 MS-SQL Server Worm," Cert.org. [Online] Available: http://www.cert.org/historical/advisories/ca-2003-04.cfm. [Accessed January 2015].

7. CERT. (August 2003). "CERT Advisory CA-2003-20 w32/blaster Worm," Cert.org. [Online] Available: www.cert.org/advisories/CA-2003-20.html. [Accessed January 2015].

8. G. Chamales. "The Honeywall CD-ROM," *IEEE Security & Privacy* , vol. 2, no. 2, pp.77-79, March-April 2004.

9. *The CIP Networks Library Volume 2: EtherNet/IP Adaptation of CIP*. Open DeviceNet Vendor Association (ODVA) & ControlNet International, Ltd, 2007.

10. C-SPAN. (November 20, 2014). "Cybersecurity Threats," C-Span.org. [Online] Available http://http://www.c-span.org/video/?322853-1/hearing-cybersecurity-threats. [Accessed January 2015].

11. U.S. Department of Homeland Security. (February 2014). "ICS-CERT Year-in-Review – 2013," Ics-cert.us-cert.gov. [Online] Available https://ics-cert.us-cert.gov/sites/default/files/documents/Year_In_Review_FY2013_Final.pdf. [Accessed January 2015].

12. S. Dunlap. "Timing-Based Side Channel Analysis in the Industrial Control System Environment," M.S. Thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, 2013.

13. *EtherNet/IP Communication Module for CP1L/CP1H PLCs CP1W-EIP61.* OMRON Industrial Automation, 2011. [Online] Available: http://www.omron-ap.com/data_pdf/cat/r37i.pdf?id=3332. [Accessed December 2014].

14. N. Hrvoje. (2011). *GNU Wget*. [Online] Available: http://www.gnu.org/software/wget. [Accessed December 2014].

15. Idaho National Laboratory. (July, 2014). "INL's SCADA Test Bed," Inl.gov. [Online] Available http://www4vip.inl.gov/research/national-supervisory-control-and-data-acquisition-test-bed. [Accessed November 2014].

16. R. Jaromin. "Emulation of Industrial Control Field Device Protocols," M.S. Thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, March 2013. (www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA582482)

17. S. Hilt. "enip-info NSE Script," NSE Documentation Portal. [Online] Available http://nmap.org/nsedoc/scripts/enip-info.html. [Accessed December 2014].

18. G. Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning.* Insecure. 2009.

19. J. Matherly. "SHODAN Computer Search Engine," Shodanhq.com [Online] Available http://www.shodanhq.com. [Accessed October 2014].

20. D. Moore, et al. "Code-Red: A Case Study on The Spread and Victims of an Internet Worm," *Proc. of the 2nd ACM Internet Measurement Workshop*, pp. 273284. ACM Press, November 2002.

21. D. Moore et al. "Inside the Slammer Worm," *IEEE Security & Privacy*, vol. 1, no. 4, pp. 33-39, July-August, 2003.

22. V. Pothamsetty, M. Franz. (July, 2005). "SCADA HoneyNet Project: Building Honeypots for Industrial Networks," [Online] Available: http://scadahoneynet.sourceforge.net. [Accessed September 2014].

23. N. Provos. (2002) *OpenBSD System Manager's Manual-honeyd(8).* [Online] Available: http://http://www.citi.umich.edu/u/provos/honeyd/honeyd-man.pdf. [Accessed September 2014].

24. N. Provos. "A Virtual Honeypot Framework," in *Proc. 13th USENIX Security Symp.*, San Diego, CA. The USENIX Association. August 2004.

25. N. Provos and H. Thorsten. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Pearson Education, 2007.

26. L. Rist, et al. (2013) "CONPOT ICS/SCADA Honeypot," Conpot.org. [Online] Available http://conpot.org. [Accessed January 2015].

27. *Rockwell Automation Publication 1756-td001h-en-p:1756 ControlLogix Controllers*. Rockwell Automation, 2013. [Online] Available http://literature.rockwellautomation.com/idc/groups/literature/documents/td/1756-td001_-en-p.pdf). [Accessed December 2014].

28. L. Shaw. (August, 2009). "A String Replacement Function, Replace str(), for the C Programming Language," [Online] Available http://creativeandcritical.net/str-replace-c. [Accessed September 2014].

29. E. Skoudis and T. Liston. *Counterhack Reloaded: A Step-by-step Guide to Computer Attacks and Effective Defenses.* Prentice Hall Press, 2005.

30. L. Spitzner. "The Honeynet Project: Trapping the Hackers," *IEEE Security & Privacy*, vol. 1, no. 2, pp. 15-23, March-April, 2003.

31. L. Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley, 2003.

32. S. Wade. "SCADA Honeynets: The Attractiveness of Honeypots as Critical Infrastructure Security Tools for the Detection and Analysis of Advanced Threats," M.S. Thesis, Iowa State University, 2011.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
26–03–2015	Master's Thesis	Sept 2013 — March 2015

4. TITLE AND SUBTITLE

Constructing Cost-Effective and Targetable
ICS Honeypots Suited for Production Networks

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Winn, Michael, M. MAJ, USA

5d. PROJECT NUMBER

15G216C

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-15-M-045

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of Homeland Security ICS-CERT
POC: Neil Hershfield, DHS ICS-CERT Technical Lead
ATTN: NPPD/CSC/NCSD/US-CERT Mailstop: 0635,
245 Murray Lane, SW, Bldg 410, Washington, DC 20528
Email: ics-cert@dhs.gov phone: 1-877-776-7585

10. SPONSOR/MONITOR'S ACRONYM(S)

DHS ICS-CERT

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Honeypots are a technique that can mitigate the risk of cyber threats. Effective honeypots are authentic and targetable, and their design and implementation must accommodate risk tolerance and financial constraints. The proprietary, and often expensive, hardware and software used by Industrial Control System (ICS) devices creates the challenging problem of building a flexible, economical, and scalable honeypot. This research extends Honeyd into Honeyd+, making it possible to use the proxy feature to create multiple high interaction honeypots with a single Programmable Logic Controller (PLC). Honeyd+ is tested with a network of 75 decoy PLCs, and the interactions with the decoys are compared to a physical PLC to test for authenticity. The performance test evaluates the impact of multiple simultaneous connections to the PLC. The functional test is successful in all cases. The performance test demonstrated that the PLC is a limiting factor, and that introducing Honeyd+ has a marginal impact on performance. Notable findings are that the Raspberry Pi is the preferred hosting platform, and more than five simultaneous connections were not optimal.

15. SUBJECT TERMS

Industrial Control Systems, Honeypot, Cyberspace Security

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			LTC Mason J. Rice, PhD, AFIT/ENG
U	U	U	U	89	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x4620; mason.rice@afit.edu